POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Digital Identity Systems for Healthcare Applications

Relatori
prof. Antonio Lioy
ing. Andrea Atzeni

Giuseppe Burrai

Anno accademico 2018-2019

Summary

In the last years, cross-border movements of people, goods and services have increased exponentially in Europe. To ease those activities and satisfy the new needs born from them, the EU Commission designs different plans taking advantage of Internet Network and digital innovation for the benefit of the European citizens. For this purpose, the CEF Digital Service Infrastructure (DSI) was set up by the Commission to finance different pan-European digital services in distinct fields (e.g., eHealth, Cybersecurity, e-Justice, e-Procurement).

In the healthcare environment, the OpenNCP platform, which is part of the eHealth DSI project, aims to improve the QoS of European patients allowing the cross-border exchange of medical data exploiting the already existent digital healthcare systems of each Member State. Through OpenNCP, an EU citizen can ask for medical treatment in another EU country, without worrying about prescriptions and his medical history: OpenNCP can retrieve this information in compliance with Member States' laws, ensuring trust and security, relying on the national health information systems. The National Contact Point (NCP), i.e. a single border node, ensures the interoperability between the different national system and links each internal domain acting as an entry/exit point for the national infrastructures. Although OpenNCP disposes of appropriate security measures for ensuring the security of the communications, it suffers from a weak patient authentication that must be performed before retrieve the medical data; instead, the authentication of healthcare professionals is under the responsibility of the country where the treatment is required.

Considering the electronic identification environment, the EU Commission provides eIDAS, i.e. Electronic Identification Authentication and Trust Services, for improving trust and confidence in the European cross-border transactions. Technically, eIDAS is another community infrastructure that ensures interoperability between the different national identification schemes (eIDs) of the European States, so that European citizens can access and use online public services in other EU countries more safely and reliably, employing their digital identity issued by their country of residence. In essence, eIDAS provides a technological solution for extending the use of the digital identity of EU citizen, released in his residence country, in some other EU Member States. For instance, this means that an Italian citizen can use public services abroad within the EU, exploiting his digital identity issued by an Italian Identity Provider.

Taking in account the properties and characteristics of the two mentioned infrastructures, the HEALTH-eID (HeID) project, in which the candidate is involved during the thesis' experience, aims to improve the quality of health services provided to EU people, integrating the eIDAS cross-border authentication with the OpenNCP data exchange service and exploiting the digital identity of the foreign citizen in order to solve the problem of the patient authentication observed in OpenNCP: in this way, a citizen abroad, which asks for a medical treatment and wants take advantages from the OpenNCP services, can authenticate himself in his country of origin via eIDAS before consenting to the transfer of his medical data through OpenNCP.

This thesis' work describes the HEALTH-eID approach to solve the OpenNCP patient authentication issue: the solution proposal consists of the development of a Web Application, the HeID connector, which integrates eIDAS in the OpenNCP workflow, in order to increase trust and security through the implementation of a reliable authentication service. Besides, the HeID project includes also a mechanism for the consensus management providing an informative for the involved patient and giving him the possibility to acknowledge or deny the consent, in agreement with the General Data Protection Regulation. Initially, the thesis' activity starts with the study of the OpenNCP platform and the eIDAS technical infrastructure for understanding the working of two systems and obtaining the technical knowledge necessary to participate at the European HeID project.

Completed the study of the technical background, the candidate actively participated at the HEALTH-eID project, a European project which took place under the supervision of the European Commission and involved several organisations belonging to different Member States (MS): in particular the project included the participation of *Servicos Partilhados Do Ministèrio Da Saúde* (SPMS) and *Caixa Mágica Software* (CMS) representing the Portuguese MS, the *Aristotle University of Thessaloniki* (AUTh) representing Greece, and finally *Azienda Regionale per l'Innovazione e gli Acquisti* (ARIA, former LISPA) with *Politecnico di Torino* (PoliTo) for representing the Italian MS. During the thesis' activity, the candidate was involved in the technical design and implementation of the solution proposal, and in details, he contributed to realise the following sub-components:

- **Workflow Manager**: Internal module of the HeID connector, it was implemented by the candidate autonomously. This sub-component manages the business and internal logic of the HeID Connector orchestrating the whole workflow.
- **Patient I/O**: Internal module of the HeID connector, it was implemented by the candidate in collaboration with SPMS. This sub-component is responsible for providing the interactive views for the patient and includes a Notification module necessary for sending emails.
- **eIDAS HProxy**: Internal module of the HeID connector, it was implemented by the candidate in collaboration with AUTh. This sub-component is responsible for managing the interactions with eIDAS, creating the eIDAS authentication request and handling the corresponding response.

The thesis work continues with the testing of the HeID connector designed and realised during the HeID project, in particular, the candidate verifies the correct behaviour of the modules responsible for patient interaction (Patient I/O) and the module responsible for interacting with eIDAS (eIDAS HProxy) in a local environment through a docker infrastructure provided by the TORSEC group of Politecnico di Torino. Additional tests about the healthcare professional interactions and the integration with the OpenNCP platform were performed by the SPMS and CMS partners, principally because they are directly involved in the implementation of those aspects.

In conclusion, the result achieved by this thesis was to demonstrate the possibility of integrating the European eIDAS infrastructure in the OpenNCP platform and proposing the HeID solution as an answer for the missing patient authentication in the current OpenNCP implementation. To this end, the "Health-eID Transferathon" event was scheduled for the 28-30 October 2019 in Porto, during which the involved partners will provide a practical demonstration of the HeID connector to the interested Member States, with a set of practical examples concerning the OpenNCP-eIDAS integration.

Contents

Chapter 1

Introduction

"I'm Italian, and I work in Greece. I want to use Greek health services if I need them. Some years ago, I worked in Portugal: here I had received several medical treatments. Before thinking about therapy, the doctors asked me for some health information, but I didn't remember exactly my clinical history.

So Portuguese and Italian doctors had exchanged emails, with attached my medical archives containing pathologies, and other sensitive data. I care about my data, and I think that emails are not an excellent way to exchange them.

Anyway, after this exchange, Portuguese doctors spent much time to understand my Patient Summary written in Italian, and I waited a long time before having a diagnosis with the corresponding therapy.

Internet technology is powerful nowadays. Is there a way to recover clinical summary through the internet, safely and reliably, respecting the patient's privacy and with the possibility of demonstrating his proper identity before recovering medical data?"

Technological progress and Globalization are two phenomena that significantly changed the European Continent and all the World. In the last three decades, the commercial escalation between different States and the increase of cross-border investments, rather than the speed of communications and information exchange, has led people travelling for business in all the Europe. To support this mobility, in 1985 some European States belonging to EEC (European Economic Community) drafted the Schengen Agreement, that "entitles every EU citizen to travel, work and live in any EU country without special formalities" [1]. Gradually more and more States decided to join this programme until the Schengen Area was born in 1995: within this special geographic Area, every person who legally belongs to one of the signatory Nations is free to move, without stopping at borders to show the passport and being subjected to boundaries checks. This political measure encouraged not only businessman and workers mobility but also the movement of all European people who have begun to travel abroad for study or leisure reasons.

Meanwhile, the EEC has become European Community (EC) in 1992 and then European Union (EU) in 2007, without changing the integration policy and persisting with the European unification program to simplify exchanges and movements between the Member States. Nowadays the Internet spread and the birth of digital world feed this mobility, leading the way to new challenges and opportunities in an increasingly connected world; but it also exposes the society to new risks (e.g. privacy). Indeed the internet and digital technologies are changing our world, making it faster and faster; thus to take advantages of the Internet Network and Digital Age, the EU Commission and Parliament have issued new directives and regulations in order to create a Digital Single Market (DSM) [2], i.e., a single market restricted to the large borders of the European Union where the free movement of products, people, services and capital is guaranteed and where citizens and businesses can easily have access to online goods and services with a high level of consumer and personal data protection, irrespective of their nationality or place of residence. Therefore, it is not difficult to observe that cross-border activities have increased throughout Europe, and consequently foreign people have started to use many public services in the other Member States, more often than in the past.

Nevertheless, DSM is only the first step towards a higher and greater EU ambition: create an inclusive Digital Society [3], in which all the digital services offered by a Country become accessible to all EU citizen. Hence, it is necessary to identify each person to assess whether he has the right to benefit from services offered by the host country, or not. In this scenario, the eIDAS (electronic IDentification, Authentication and trust Services) Regulation and Electronic Identification Systems plays a fundamental role, in particular the Digital Identity becomes a necessary tool in the citizen's hands to safely use the digital services that each State can offer, like eGovernment, eHealth and other Digital Public Services.

Considering the increase in the usage of information technology in the healthcare sector, if we think of what doctors can do today relying on telematics technology, asking health information about their patients from colleagues that work in other Countries while healthcare professionals using more often ICT applications for different medical purposes, the European Union decided to define a way for regulating and giving common guidelines for enhancing the use digital information systems in healthcare. Thus is born the *eHealth* word, which expresses the use of ICT for providing digital health and care services. According to the Commission, eHealth is:

> "Digital health and care refers to tools and services that use information and communication technologies (ICTs) to improve prevention, diagnosis, treatment, monitoring and management of health and lifestyle. Digital health and care has the potential to innovate and improve access to care, quality of care, and to increase the overall efficiency of the health sector" [4].

In particular for guaranteeing better healthcare experience in the European Union, the Commission establishes, with the *Directive 2011/24/EU for patients' rights in cross-border healthcare* [5], that each EU citizen has the right to access to public health services in any EU country and to be reimbursed for the treatment abroad by his residence country. However, for improving the quality of service, it is necessary to interconnect the National healthcare systems in order to exchange information about patients abroad concerning the most important aspects of their health, such as allergies, surgeries, current medication and other clinical records. For the purpose, the EU Commission proposes the realisation of digital cross-border infrastructure through the eHealth Digital Service Infrastructure (eHDSI) [6] programme, which foresees the integration of national platform at the community level for exchanging Patient Summaries and ePrescriptions. The National Contact Point component ensures this integration: it is deployed at the national country level and is responsible for the interoperability of the national health system of each country among the other European healthcare systems. From a technical point of view, the digital eHDSI infrastructure is based on the OpenNCP platform, which aims to retrieve the clinical records of a patient that requires for treatment abroad, ensuring the security of the transmitted data. Although appropriate security measures ensure the security of the communication, OpenNCP suffers from a weak patient authentication: indeed, while the authentication of the healthcare professional (HP) is under the responsibility of the country in which the citizen asks for a service, the patient's one is done locally by the HP checking the patient's health insurance card, and is practically absent in the home country of the involved subject, i.e. the country in which the data are stored. Hence, in summary, the OpenNCP platform, at state of the art, retrieves some sensitive data, through a cross-border exchange, without performing the authentication of data owner in the entity where the data resides. In response to this problem, comes the proposal to integrate the eIDAS cross-border authentication service in OpenNCP, in order to improve security providing a reliable authentication mechanism for the patient abroad before authorise the cross-border data exchange: the technological solution, designed and developed within the HEALTHeID project, consist in the development of a software NCP-eIDAS connector, called HeID Connector, which gives an active role to the patient handling his authentication process relying on eIDAS and, based on the result of the cross-border authentication, allows, or not, the recovery of medical data through OpenNCP. Also, in its workflow, the HeID Connector foresees the informative and consensus management, designed according to the EU General Data Protection Regulation.

This document reports the fundamental principles of the eIDAS and GDPR Regulation with an overview of the eIDAS and OpenNCP infrastructures; the description continues presenting the HeID solution proposal for solving the weaknesses of patient authentication, followed by the design choices and the implementation details. The thesis activity took place within the European HEALTH-eID project, which involved directly the European Commission and various organisations representing the Member States participating in the project: namely Servicos Partilhados Do Ministèrio Da Saúde (SPMS) and Caixa Mágica Software (CMS) for Portugal, Aristotle University of Thessaloniki (AUTh) for Greece, and ARIA (former LISPA) with Politecnico di Torino for Italy. The design activity, such as the implementation, was carried out by the candidate with a strong and active collaboration among the mentioned partners.

In conclusion, the thesis' project presented here intends to offer a concrete solution to the problem mentioned above, exploiting the eIDAS interoperability services in the field of digital identities, for improve trust and security inside the eHDSI OpenNCP.

Chapter 2

Background

This chapter presents some key concepts understand what a Digital Identity is and how it works. It includes also a brief overview concerning the security property of information systems and the General Data Protection Regulation.

2.1 Security properties of Information Systems

Computer security is the set of means and technologies used to protect Information Technology (IT) systems in terms of availability, confidentiality and integrity of IT goods or assets. The NIST [7] agency describes computer security in his Computer Security Handbook as follow:

> The protection afforded to an authentication information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information systems resources (includes hardware, software, firmware, information/-data, and telecommunications) [8].

A security service is a service realised to ensure a specific kind of protection to digital resources which reside or are exchanged by information systems. The RFC-4949 provides the following definition: "A processing or communication service that is provided by a system to give a specific kind of protection to system resources" [9]. For realise a security service, it is necessary to design security mechanisms that guarantee the following security properties [10]:

- **Authentication:** the assurance that the parties involved in a communication are the ones that claim to be, and is called peer entity authentication. Another aspect of satisfying is the data-origin authentication, which means having to guarantee the identity of the source of the received data.
- **Access Control:** it is the capability to restrict access by using authentication or authorisation mechanisms in order to control the entities attempting to access the system and prevent the unauthorised use of a resource.
- **Data Confidentiality:** the protection of transmitted data from passive attacks. Several levels of protection can be applied to protect the content of data, depending on the level of the protocol stack to protect. Another aspect of confidentiality regards the protection of information that can be deduced from a traffic analysis (e.g. source and destination, frequency, length).
- **Data integrity:** the certainty that the accepted data are exactly the same as transmitted by an authorised entity (i.e. contains no alterations).
- **Nonrepudiation:** it avoids that a sender/receiver may deny having sent/receive a message; it protects messages proving that the entities involved in a communication are in fact participating.

- **Availability Service:** the property of a service or a resource to be available when some authorised entity requires it.

An information system must satisfy the mentioned properties realising the corresponding security mechanisms, which can be designed for providing security measures at different levels.

2.2 Federated Identity Systems

The internet widespread has led to growing up the number of online services. Each service, to protect itself from malicious users, has to implement some security policies including an authentication service with an access control mechanism, in order to define an identity for the entities attempting to use the services, providing them with credentials, e.g. username and password, to access the service in the specific domain. In this scenario, an ordinary user that wants to read his email needs to authenticate first; the same if he wants to read an online newspaper or manage his pay-TV account. However, each online service belongs to a different domain; therefore, the user has several access codes in his hands where each code is specific to a service. Having a lot of different specific credentials increases the probability of identity thefts and unauthorised accesses. Federated Identity allows to have a single credential to access different resources belonging to different security domains and reduces the insecurity due to the high number of access codes managed by the user, also increasing his awareness: indeed, the comfort to use a single digital identity to access online services makes its credentials important and precious for him, because they represent the unique key that he can use to do online activities, considering that some of these are strictly personal, e.g. Internet Banking or emails checking.

2.2.1 Entities

In each domain, a single accredited entity manages credentials, i.e. Identity Provider, which guarantees that the user is who he claims to be. In a federated system, organisations and service providers must agree and trust each other, establishing rules and guidelines for managing these relationships of trust; these agreements are called *federation*. In other words, a *federation* is a heterogeneous system composed of different organisations and domains, in which each participant must sign a set of rules that allow all members to trust each other, and where multiple entities can interact through a shared infrastructure. Those entities fulfil different actions in each domain and can act as a Service Provider (SP), an Identity Provider (IdP) or an Attribute Provider (AP).

Service Provider (SP)

Service Provider is the entity responsible for providing the requested service to one or more members of the federation; members must be authenticated.

Identity Provider (IdP)

Identity Provider is the entity responsible for maintaining and managing information relating to the identity of the End User. IdP has the task to authenticate the user and guarantees for his identity.

Attribute Provider (AP)

Attribute Provider is an entity which gives support to IdP if the authentication process needs of some additional attributes that qualify users (states, roles, titles, positions).

2.2.2 Digital Identity and Federated Identity

Digital Identity is the set of attributes that a natural or legal person has in a given domain. A person can have *multiple digital identities* at different services such as social networks, e-commerce sites or banks. For users, managing this multiplicity is very complex and can significantly increase security risks such as identity theft or privacy breaches.

Federated Identity is an extension of Digital Identity to multiple security domains: in essence, the users have a *single digital identity* and can be authenticated a single time before accessing to different applications or resources that belong to different domains.

2.2.3 Federated Identity Management

Overall, for sharing a unique digital identity accepted by multiple domains, the involved organisations must form a federation by agreeing on the standards and technologies to be used for ensuring the portability of digital identities, and define a mutual level of trust to exchange the information securely: the set of these accepted policies, technologies and standards outlines the Federated Identity management. With the Federated Identity, the users can access to different services belonging to heterogeneous domains, authenticating himself by an organisation, which must belong to a federation and emit a security ticket that can be employed by the user as authorisation proof to access services related to other domains and managed by organisations which are joined the federation.

In particular, to authorise a user belonging to another domain, the service provider of the involved company needs an identification assertion from one of the IdPs which belongs to the federation, including the necessary attributes required for providing the requested service. To allow the user authentication process, the Service Provider displays all IdP profiles registered within the federation, relying on a register that contains the list of trusted IdPs: the user chooses the authentication point from the list, so then the SP redirects him to the selected IdP, where the authentication process begins. Once the authentication assertion is complete, with the set of necessary attributes, the IdP redirects the user to the SP that initially started the authentication request. Afterwards, the authentication assertion is received by the SP, which can complete the authorisation process and allow access to the service initially requested.

2.3 Reference technologies

2.3.1 The SAML Standard

The *Security Assertion Markup Language* (SAML) is an OASIS [11] standard used for exchanging authentication and authorisation data in a multi-domain distributed system through the so called *assertion* that are security object which contains security information about the authenticated entity. The SAML Standard aims to provide an XML-based syntax, and related rules, for exchanging security information between remote entities that interacts on the internet network. This information consists of assertions exchanged between *asserting parties*, i.e. the entities that issue them, and *relying parties*, i.e. the entities that request them and that can make use of them for authentication and authorisation purposes.

Currently, the SAML specification is at the version 2.0, that is incompatible with the previous versions, and has a modular organisation, expressed by the following concepts:

- **Assertions:** SAML objects which contains information about a subject or an entity. They are used to transport safely authentication and authorisation information. Their structure is defined in the SAML-core document [12, section 2].
- **Protocols:** The assertions need to be appropriately exchanged for communicating the authentication attributes at the entities which requires them. Therefore, the SAML Standard defines the related protocols which must be used in a SAML information exchange, which

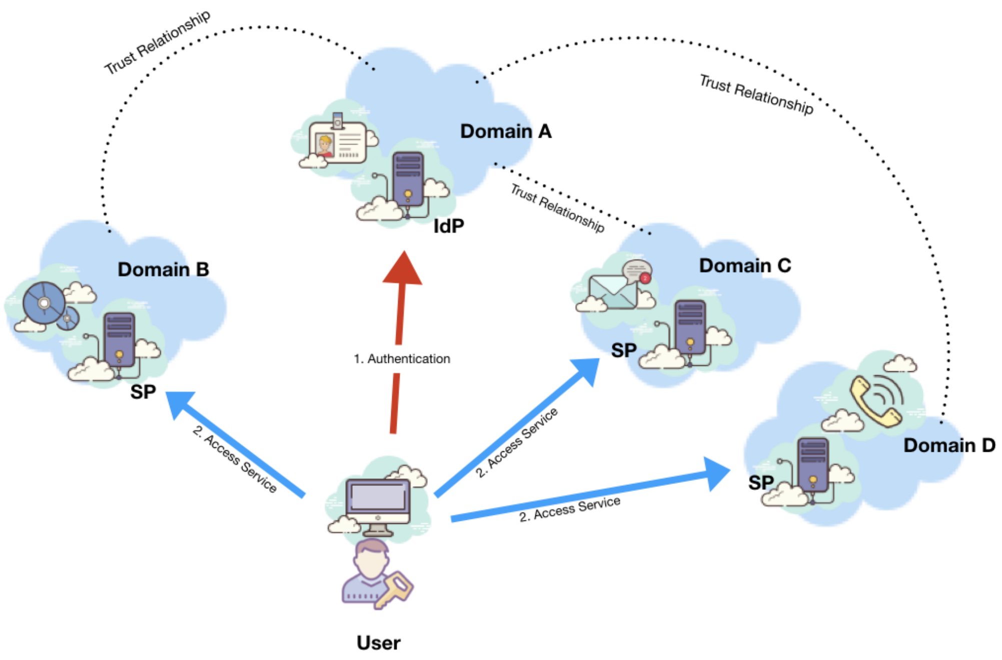


Figure 2.1. Federated Identity Architecture: the user is authenticated once before use application services belonging to domains of different organisations that have joined the federation.

consists in a request-response mechanism. The details of such protocols are explained in [12, section 3].

- **Binding:** The protocols establish the structure of the information that can be exchanged between the involved entities, but do not determine the specific modes of transport. About that, the the SAML Standard defines specific bindings for indicating how to effectively implement in SAML the exchange of security information through certain transport protocols (for example HTTP or SOAP). More information is detailed in the SAML-Bindings document [13].

- **Profiles:** a profile identifies a particular combination of types of assertions, protocols and SAML bindings designed to support a specific use case considered relevant for online interaction between entities. The most used profile is the Single Sign-On (SSO), defined in [14, section 4.1], which describe how SP can make available services usable via web browser by users,

Furthermore, the SAML specification foresees the possibility to extend the standard (e.g. adding other types of assertion), as well as the use of metadata [15], i.e. constructs having a standardised structure for the description of some information that characterise SAML entities and support interactions between them. In particular, the metadata describes the properties of the entity with the necessary information useful to correct initialise a SAML session. Usually metadata are defined in a XML document which contains the informations expressed within a `<EntityDescriptor>` XML element: before any interaction, each entity must have the counterpart's metadata available for starting a SAML session, therefore, the involved entities needs to publish and make its metadata available and remotely accessible through an exposed interface that consist of an HTTPS service, whose address is well-known and easily recoverable.

2.3.2 Transport Layer Security

Transport Layer Security (TLS) [16] is a standard technology that provides a security service ensuring the security of connections in an internet communication and protecting the data exchanged between the involved entities. TLS is an updated and more secure version of the deprecated *Secure Socket Layer* (SSL), designed by *Netscape Communications*: this protocol provides security at Transport Layer, establishing a end-to-end communication between the source and the destination on a TPC/IP network, and in particular it ensures authentication (mandatory for a Web Server), encryption and data integrity exploiting different cryptographic methods and algorithms that can be applied on the messages before sending them over the network [17]. Indeed with SSL/TLS, two entities that want to communicate with each other must, first of all, negotiate how to create a secure channel and which security parameters to use, i.e. the so called "`cipher suite`". Second, they proceed with the x509 Certificate exchange, which ensures the identity of the involved parts and provides the cryptographic keys for ciphering the messages: at this point, each part can verify the authenticity of the counterpart's certificate in order to check its validity and establish a trust relationship. Finally, a secure channel is created and the two entities can exchange message in security. This mechanism is called *SSL/TLS Handshake* and is described in details in [18], [19].

The *HyperText Transfer Protocol Over SSL* (HTTPS) [21], is the most used protocol for establish a security channel combining SSL/TLS with the HTTP standard. Indeed HTTPS is a variant of the HTTP protocol used for transferring web pages on the internet; but while in HTTP the communications are all in plaintext, the HTTPS version provides a safe communication creating a secure channel, encrypted through the SSL technology, in order to prevent communication with untrusted systems and avoid the manipulation of data of a potential malicious user.

2.4 General Data Protection Regulation

The *General Data Protection Regulation* (GDPR) [22] is a European Regulation which enforces and normalises the data protection of European citizens; it is entered into force since the 25 May 2018. The GDPR only regulates the processing of personal data of natural persons and therefore the personal and similar information of subjects such as corporations, companies and public bodies, associations and foundations, are excluded from the application of the rules expressed in the document. Its application is extended to organisations that have their registered offices outside the EU, but that process the personal data of residents in the European Union. Its purpose concerns three basic objectives:

1. protect people about the processing of their data
2. give at EU citizens more control in the way of their data are stored and treated,
3. check how personal data are exported outside the UE.

Hence, public administration and private Service Providers, which treats with some personal information about their users, must comply with the requirements and tools foreseen by the Regulation that can be summarised as follows:

- The **treatment register** (Art. 30) is a priority fulfilment for keeping track of treatment activities, which the Data Controller is obliged to comply according to the principle of "Accountability". Some activities must be tracked in this register, for instance, the purpose of data processing, the description of data processed, the data protection measures adopted, the description of the categories of the data subject and others. This register constitutes the more important document to prove that the Data Controller has fulfilled the directives of the Regulation.
- **Risk assessment** (Art. 32): according to the mentioned article, the owner or manager of the processing of personal data must put into practice technical and organisational measures,

so that the level of security is adequate to the risk. For this reason, it is necessary a model that assesses what the risks to privacy are, what is the security level, also calculating the probability of occurrence of threats, and what are the consequences that can derive from a damaging event on the data.

- The **impact assessment** (Art. 35) is a process aimed at identifying, describing and evaluating an impact assessment on data protection if a type of treatment may present a high risk for the rights and freedoms of natural persons to whom personal they data refer. The impact assessment is technically defined as "Data Protection Impact Assessment" (DPIA), and the Article 35 of GDPR gives the guidelines on how to perform this evaluation, which should be done in advance, before the data processing. The DPIA process can occur, in particular, when the use of new technologies is involved; when the impact assessment indicates that the treatment presents a high risk, the owner is obliged to consult the supervisory authority before proceeding with the treatment.
- The **prior consultation** (Art. 36) is the procedure by which the data controller, before to proceed with the processing, consults the supervisory authority, e.g. for Italy the"Garante della privacy", if the performed impact assessment indicates that the treatment would have a high risk in the absence of measures, which the data controller can take to mitigate the risk.
- The **Data Breach procedure** (Articles 33 and 34) exposes the action plan that the organisations must follow if a data breach will hit them. The affected organisations must communicate the data breach to the Guarantor through a standard notification procedure reporting the situational analysis, evaluation of data involved, impact description, preventive measures and actions, supervision. Besides, the organisations involved in a data breach must notify it within 72 hours of its discovery.
- The **agreement with joint owners** (Art. 26) is a model which try to solve some problems that revolve around the concept of "responsibility" in the field of personal data protection for concretely determining "who does what" and what impacts the individual choices have in a particular situation. The model agreement establishes the rights and obligations of each joint owner about the processing of personal data.
- The **appointment of the managers** (Art. 28) it is a mandatory appointment that gives the manager a large number of tasks and charges, that may imply, among the other things, the keeping of a treatment register or the implementation of security measures and also requires a mandatory signing of a specific contract. This normative defines general guidelines in order to leave to parties a significant autonomy in the definition of the task.
- The **appointment of the authorised** (Article 29) and the training for the authorised (Article 39): anyone who acts under the authority of the Data Processor and who has access to personal data, cannot process such data unless authorised to do so by the data controller. Moreover, adequate training is required both for those who coordinate the treatments and the authorised to treat the data: the staff should include persons in charge (or managers) of the data processing, but also managers and developers that work in IT field and directors of commercial units. The training, in addition to being a "prerequisite", should be aimed at illustrating the general and specific risks of data processing, the organisational, technical and IT measures adopted, as well as the responsibilities and sanctions.
- **Management of consents and information** (Articles 13,14,6,7,8,9). The mentioned articles define how the consent has to be provided and managed by a Service Provider that processes personal data. In summary, the consent must be:

  1. `Unequivocal.`
     There must be no doubt that the data subject has given his consent. Therefore the concerned person must perform a "positive action", such as, for example, ticking a box.
  2. `Free.`
     The interested party must be in a position to give his consent freely, without any possibility of having to suffer "negative consequences" following the failure to grant

consent. For instance, the service's provision or the contract's execution must not "condition" the involved person to give unnecessary consent to the provision of the service or the performance of the contract.

3. `Specific.`

   Consent is strictly related to the purpose for which that processing is performed. If the collection and processing of data have multiple purposes, consent must be given for each different purpose.

4. `Informed.`

   The interested party must know what data are processed, and for what purposes they are treated, how they are treated, his rights and the effects of his consent. The interested party must be informed through a specific Information that, according to the provisions of the GDPR, must be drafted in a transparent manner and with simple and understandable contents.

5. `Verifiable.`

   The data controller must be able to demonstrate that the data subject has consented to the processing of the data, distinguishing the different purposes for which it was provided.

6. `Revocable.`

   The interested party must be able to revoke the consent at any time, with the same ease with which he has given it.

- The **appointment of the DPO** (Articles 37, 38, 39), i.e. the Data Protection Officer. DPO is a specialist figure, who is appointed by the Data Controller, whose task is to support the Data Controller in the application of the procedures concerning the new Regulation, also acting as an intermediary between the Control Authorities and those directly concerned. The designation of the DPO is mandatory only in particular situation, such as large-scale data processing, or commanded by a Public Authority, or concerning particular categories of sensitive data. In all the other cases, it is optional but recommended to facilitate the application of the rules.

- **Data transfer abroad** (Articles 44 to 50). The Regulation also defines the instruments through which the EU laws, concerning the protection of personal data, interact with the rest of the world. Indeed cross-border data transfers outside the EU are generally prohibited unless under specific guarantees. The article 44 of the GDPR is explicit in establishing that transfers of personal data outside the European borders are allowed only in certain circumstances; besides, the recipients of data are divided into two categories: other Extra-UE countries and international organisations, such as United Nations, UNESCO, and so on.

- **The codes of conduct** (Articles 40 and 41) are encouraged by the Regulation: they are rules of conduct or uniform practices developed by various international bodies or even by individual States, particularly widespread in international economic relations. The Regulation treats these codes from a community perspective. The mentioned articles focus on the need to encourage the drafting of codes of conduct, the establishment of data protection certification mechanisms and data protection seals and marks.

Also, the Regulation clarifies and adequately defines some principles regarding the privacy of data, which are the `privacy by default`, the `privacy by design` and the `Necessity & Proportionality of data`. The former consists in the obligation to have, regardless (by default), a well-defined procedure for acquiring, processing, protecting and disseminating data that a Service Provider collects so that it can guarantee the respect of privacy regarding sensitive data. The second principle foresees that during the "designing" of a new business process, the part concerning data protection must always be taken into consideration. The latter regards the data which a Service Provider can ask a user, and in essence, it is inspired to the "need-to-know" principle: it is established that the data acquired and processed must only be those strictly necessary for the purpose.

During the thesis' work, special attention was paid to the GDPR, especially for the informative to the patient, the consensus management and the necessary personal data to store in the RDBMS database.

Chapter 3

Electronic IDentification, Authentication and trust Services

This chapter presents the European eIDAS regulation and the main legal norms underlying it. Then it continues with an overview of the eIDAS technical infrastructure and the functional blocks that compose it, reporting an example of an authentication flow.

3.1 eIDAS Regulation

The EU Commission identifies trust and security processes as the most critical factors in the cross-border interactions. To increase digital confidence in businesses, public administrations and service providers that operate in different economic areas - such as the banking and financial sector, health, public administration and transport, the European Union issue the eIDAS EU Regulation 910/2014 (electronic identification and trust services for electronic transactions in the internal market [23]).

Juridically, the eIDAS Regulation repeals the Electronic Signatures Directive 1999/93/EC [24] exceeding every rule at the national level in order to promote greater interoperability. For definition:

> "A directive [25, art.288, comma 3] shall be binding, as to the result to be achieved, upon each Member State to which it is addressed, but shall leave to the national authorities the choice of form and methods".

Hence, a directive is a legislative act that establishes goals that will have to be achieved by the EU Countries. However, each State must achieve these objectives in the ways and with the means that it will decide, in turn, with its own internal regulatory activity.

> "A regulation [25, art.288, comma 2] shall have general application. It shall be binding in its entirety and directly applicable in all Member States".

Instead, a regulation is a binding act that is directly applicable to all the subjects of the internal system (constitutional bodies, judges, public administration, citizens, companies, institutions, etc.), without any form of adaptation by the Member States: the adaptation is not only not necessary, but it is expressly forbidden. The regulations have mandatory, direct and immediate effect without having to pass through any regulatory activity of the individual States [26, Chapter 18, Section 4.3]. The use of regulations is a fundamental point to ensure interoperability between the European States.

In the field of electronic identification and trust services, eIDAS provides the legal framework for cross-border recognition and guarantees both interoperability and legal certainty for each transaction. Hence eIDAS responds to the need to have explicit guarantees on the identification of counterparties, on the legal value of the documents and their transmission and, in general, of the digital services made available within the Digital Single Market: in summary, eIDAS ensures that citizens, companies and public administrations can access and use online services in a safer, more responsible and more convenient way exploiting their national electronic identification schemes (eIDs) to access public services in other EU countries where eIDs are available. The eIDAS implementation must follow the next general principles, among others, in order to obtain trust and confidence of users in the use of cross-border digital services:

- the mutual acceptance of national eID, which is necessary for service providers to serve the requests of European people,
- a common framework for secure interaction between citizens, companies and public administration to increase confidence in the digital environment,
- the technological neutrality of requirements in order to not impose a specific technology in the individual implementations,
- the interoperability between the different digital signature solutions,
- a precise eID quality level defines the level of trust in national electronic identity, according to the technical and infrastructural controls carried out in the national infrastructure,
- country-specific supervision organisations are necessaries to verify the Regulation adoption by the service providers (e.g. for data privacy) reporting the status to data protection authorities at the European Commission.

However, digital identity is not enough to guarantee security and legal validity in the entire infrastructure. Therefore eIDAS needs Trust Services, which are defined as a set of electronic services characterised as follows:

- services for the creation, verification and validation of electronic signatures, electronic seals, electronic time validations, electronic delivery services certified, certificates relating to these services;
- creation, verification and validation of website authentication certificates;
- signature preservation services; electronic seals or electronic certificates relating to these services.

This set of services must ensure the time-stamping of documents and transaction, electronic seals, electronic delivery, legal admissibility of electronic documents and website authentication.

Figure 3.1. EU trust mark for qualified trust services (AgID).

Besides, a Trust Service is called "qualified" if it fulfils the specific requirements established by the eIDAS Regulation and provides the relative guarantees in terms of safety and quality of services; a Qualified Trust Service is subject to supervision by country-specific organisation, one per Member State, e.g. the "Agenzia per l'Italia Digitale" [28] for Italy. Furthermore, eIDAS punctually indicates the tasks and the methods of mutual assistance to establish a reference framework for supervision, it shall be as uniform as possible on the territory of the Union.

The Qualified Trust Service providers are authorised to identify its services through the use of the EU trust mark for qualified trust services: this trust mark is regulated by the Commission with the Regulation (EU) 2015/806 of 22 May 2015 [27]. It has four possible representations, shown in Figure 3.1.

3.2 eIDAS Infrastructure

The eIDAS infrastructure is technically based on STORK (Secure idenTity acrOss boRders linKed [29]), the European project completed in 2015, that implements a cross-border system which makes existing authentication models of different member states interoperable [30].

Each Member State, to be active in the eIDAS world, has to notify his national electronic identification (eID) schema, e.g. Italy notified the "Sistema Pubblico di Identità Digitale" (SPID) as its internal authentication scheme. Once examined and approved by Commission, the eID schemes will have to be recognised by all other countries participating in eIDAS, under certain conditions: indeed the Regulation introduces a three-level security system [23, Article 8, Assurance levels of electronic identification schemes], thus the obligation to recognise a person identified in another State applies only if the security level of his national identity scheme is the same (or higher).

In order to obtain a correct exchange of information among different eIDs, the EU Commission issued a set of technical specifications [31], foreseen by the Regulation, which provides details about:

the interoperability architecture that specifies in details all the components of the eIDAS Network necessary to achieve interoperability of notified eID schemes;

the SAML message format to give specification on the structure of eIDAS SAML messages (e.g. AuthnRequest, Response and Metatata messages);

the SAML attribute format that reports necessary information to map national eID attributes with eIDAS attributes correctly (e.g. SAML Attribute Naming);

the eIDAS cryptographic requirements needed to ensure confidentiality and protect the communication between eIDAS nodes employing SAML and TLS protocol.

The Figure 3.2 shows the logical diagram of the eIDAS architecture. Consider a foreign citizen, belonging to Country A, that wants to use some digital services in Country B. The citizen holds his digital identity issues by Country A with an eID-scheme different from Country B. However, both country A and country B have implemented eIDAS to ensure cross-border authentication of European citizens belonging to another Member State. Note that both States involved must have adapted the eIDAS node to make their infrastructure interoperable with eIDAS, following the specifications dictated by technical regulations. In a real scenario, a foreign citizen that requests a digital service in Country B has to authenticate himself to prove his identity before he can benefit the service; thus the authentication process flow can be summarised as follow:

1. The citizen accesses to a Country B service provider and requests to use a service. The service provider (SP), which operates in Country B, sends an authentication request toward the eIDAS-Connector accountable for its domain. eIDAS-Connector shall ask to citizen his origin country if this kind of information is missing in the request.
2. At this point, the eIDAS-Connector shall send a SAML-Request to the eIDAS-Proxy Service of Country A:

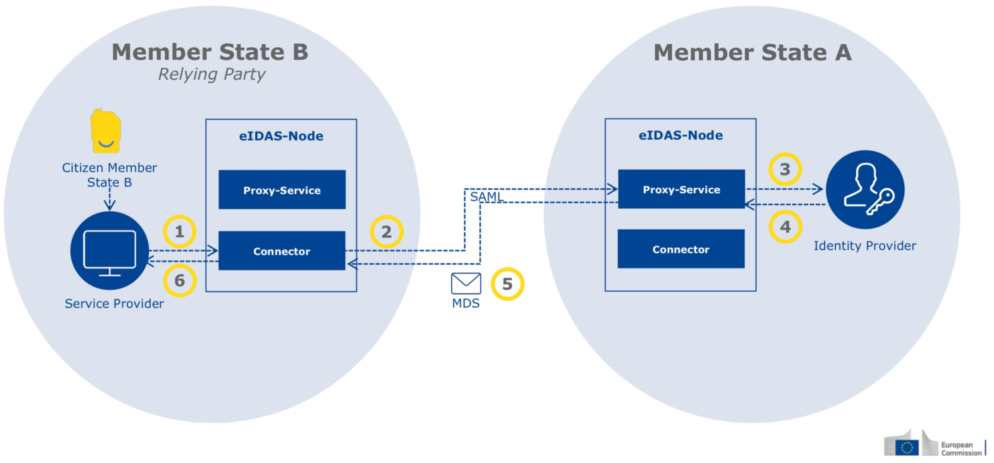


Figure 3.2. eIDAS logical cross-border architecture (eID Community).

(a) if the eIDAS-Proxy Service serves several eID schemes, the Service should provide a scheme selection interface for the user;

(b) if the requesting relying party is a private entity, the Service may reject the request if the terms of access of the eID scheme are not fulfilled;

(c) if the eIDAS-Service can not fulfil the requested (or higher) Level of Assurance, the request MUST be rejected.

3. The eIDAS-Proxy Service shall send the request to the National Identity Provider (IdP).

4. The Country A national IdP, according to the selected eID scheme at least on the requested Level of Assurance, perform the authentication of the person and shall send the SAML Response to the eIDAS Proxy Service.

5. The eIDAS proxy Service shall send a SAML Response to the requesting eIDAS-Connector containing an encrypted SAML Assertion with the eIDAS Minimum DataSet (MDS).

6. The Connector must verify that the Level of Assurance indicated in the Assertion matches or exceeds the requested Level of Assurance, and send the received authenticated person identification data to the requesting SP.

To exchange the authentication data, eIDAS employs the SAML protocol with WebSSO Profile, so error handling shall follow the SAML specification.

3.2.1 eIDAS Node

The eIDAS-Node is the core element of the same infrastructure that allows the cross-border interoperability of digital identity (eID) systems in EU member states. Each Member State has to define its technical interfaces for the eIDAS Connector and eIDAS Proxy Service in order to achieve interoperability between eIDAS node and its national eID system, in compliance with minimum specifications and requirements. The two software modules play different roles depending on the kind of request coming in the node. Please, note that eIDAS provide two integration scenarios for the eIDAS node development: Proxy-based, that implies the use of a Proxy Service in the Country A, or Middleware-based, that defines the adoption of a software module developed by Country A and installed in the eIDAS node of the Country B. In this work the focus is on the eIDAS Proxy-based solution: for further details about the two different implementations, please consult the reference documentation [31].

eIDAS Node Connector

The eIDAS Connector is an internal component of the eIDAS node, responsible for asking and providing the digital authentication of a citizen, that belongs to another Member State, from the eIDAS infrastructure. Indeed it receives the authentication request from a Service Provider, i.e. the Relying Party, and forward this request in the eIDAS world. If the patient authenticates himself correctly in his National IdP, the Connector will receive the eIDAS SAML assertion as a response, containing the SAML Attributes that certify the identity of the citizen.

eIDAS Proxy Service

The eIDAS Proxy Service is another component of the eIDAS node that receives the authentication requests coming from the eIDAS environment. The Proxy Service elaborates those requests and translates them in the National format so that it can forward them to National identity provider (IdP). After the citizen authentication, the Proxy Service receives the response from IdP, maps national attributes with eIDAS attributes and transmits the eIDAS assertion to the Relying Party located in the other Member State, providing thereby the cross-border authentication.

3.2.2 eIDAS Minimum Data-Set

The eIDAS interoperability framework allows for cross-border identification and authentication processes through the exchange of SAML 2.0 messages, including personal and technical attributes. The Regulation specifies a minimum data-set that must be supported by all the eIDAS node and establishes a set of mandatory attributes required to define the cross-border authentication appropriately and according to different legislations. The Table 3.1 shows the eIDAS SAML minimum data-set defined by the standard for a natural person: note that the Mandatory attributes are required by the Regulation; instead, the Optional attributes may be supplied by a Member State if available and acceptable to national law.

Attribute (Friendly) Name	**eIDAS MDS Attribute**	**Mandatory/Optional**
FamilyName	Current Family Name	Mandatory
FirstName	Current First Names	Mandatory
DateOfBirth	Date of Birth	Mandatory
PersonIdentifier	Unique Identifier	Mandatory
BirthName	First Names at Birth	Optional
PlaceOfBirth	Place of Birth	Optional
CurrentAddress	Current Address	Optional
Gender	Gender	Optional

Table 3.1. eIDAS minimum data-set for Natural Persons (eIDAS attribute profile).

Chapter 4

Smart Open Services for European Patients

This chapter describes the OpenNCP platform, initially born during the ePSOS project and then incorporated in the CEF eHDSI activity. It continues with a description of the OpenNCP platform and its internal component, with a particular focus on the authentication and consensus management processes. Finally, some security consideration concerning the OpenNCP platform is reported.

4.1 The epSOS project

Smart Open Services for European Patients (epSOS) project was a European pilot, begun in 2008 and co-funded by the European Union through the Competitiveness and Innovation Program (CIP)[32]. epSOS aimed to create an electronic health data exchange service in the European context, based on existing national information systems with the capability to obtain sensitive medical information securely, in compliance with the regulatory frameworks and information systems existing in the Countries participating to the initiative [1] . To realise this plan, the cooperation between all participants was one of the most important keys: indeed active cooperation is strictly necessary to achieve a common eHealth infrastructure ables to guarantee interoperability between the National eHealth systems, surely different in every Member State.

The epSOS project lifetime can be divided into two-time frames: epSOS I (2008-2010) and epSOS II (2011-2014). During the former frame, the main goal was to define the Patient Summary with the electronic pharmaceutical prescription (ePrescription), both oriented to ensure an international harmonisation of the different documents adopted by each Member State, and to develop an European Top-Level ICT Infrastructure that allows to single European healthcare systems to communicate with each other, with security and trust. In the latter, the primary topic is to provide direct access to Patient Summary, giving the possibility for citizen and Healthcare Professional, of reading medical documents in a specific language, even if different from the language in which the Summary has been compiled.

An essential challenge was to realise the project keeping unchanged the existing National eHealth infrastructures, taking into account also the differences in national legislations. Therefore, to overcome this primary constraint, it needed to create a central European system able to communicate and exchange data with the other peripheral systems: thus was born the OpenNCP platform, of which the National Contact Point (NCP) is the core element. Each NCP is related to each Participating Country, and it is also interconnected to other NCPs; basically, its purpose is

[1]Austria, Belgium, Croatia, Cyprus, Czech Republic, Estonia, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Lithuania, Luxembourg, Malta, Netherlands, Poland, Portugal, Slovenia, Spain, Sweden.

to act as a bridge between the National healthcare infrastructure and the epSOS infrastructure. OpenNCP software is designed, coded, tested and delivered by the OpenNCP Community [33], an open group of people that develop a set of Open Source Components to create a basic NCP implementation that each Country can use as a starting point to build a custom NCP, suitable for communicating with the own National health domain.

During the 5th meeting of the eHealth Network (Athens, 13 May 2014) [34], which took place at the end of the epSOS project, a sub-group of the Network was set up, in charge of preserving the developed services and pursuing the realisation of the same services on a large scale within the Connecting Europe Facility (CEF) community program [35]. CEF is an EU funding instrument that defines how the Commission can provide financial assistance to the trans-European networks in order to support infrastructure projects of common interest in three principal sectors: transports (CEF Transport), energy (CEF Energy) and telecommunications (CEF Telecom), giving priority to projects that have a European added value and significant advantages for the EU society and that do not receive adequate financing from the market. The CEF operating period started in 2014 and will end in 2020. As regards the CEF Telecom, the main areas of financing concerns the fast and ultra-fast broadband networks, to which 170 million euros have been allocated, and the Digital Service Infrastructure (DSI) that delivers different pan-European digital services for specific domains, (e.g. eHealth, Cybersecurity, e-Justice, e-Procurement), with a budget of 970 million euro, for a total of 1.13 billion euros.

Hence, the CEF programme incorporates the epSOS project inside the eHealth Digital Service Infrastructure (eHDSI), which aims to improve cross-border eHealth, providing tools and services fit for purpose. These services can be grouped in core services and generic services. The core services are set-up and disposed of by the European Commission using its resources and through calls for tender financed by CEF, while the Participating Nations fund the general services with the possibility to access to CEF community funds through a "Call for proposals". Generic services are deployed in part by each Member State and are provided by the National Contact Point for eHealth (NCPeH). Each NCP interacts with the core services, placed at the EU level, and acts as a gateway between the eHDSI infrastructure and the National infrastructure. Consequently, all cross-border communications must pass only through the NCP.

The last Call for proposals, concerning eHealth, was published on 4 July 2019 [36] with an indicative budget of 5 million euros: it aims to improve Patient Summary and ePrescription interoperability, to implement new guidelines, to conduct further testing and audit activities at the core level. Regarding the generic services, the idea is to keep the support to the technical development of National Contact Points, both for already Participating Countries and Countries that would participate right now. Moreover, over time, new features and conditions could be issued, based on the growing needs of the network, so each Country must align and comply with the new requirements.

4.2 eHDSI System Architecture

eHealth DSI is an infrastructure that employs ICT tools and services to enable the cross-border exchange of health data between European healthcare systems, in compliance with Member States' laws, in trust and security. Europe is a technologically developed continent, and most Member States already have a national infrastructure that can manage public health facilities digitally. However, the differences between the system implementations of each Country do not allow direct communications among health facilities because they have been designed differently and often with different technologies. In the European eHealth environment, the challenge was to create a top-level architecture that ensures interoperability between all the healthcare infrastructures keeping the individual national system and legislations unchanged. Note that those differences are not minimal: Europe is a set of profoundly different States from history and culture, and these aspects crucially influence how people think and do things. However, the variety of languages is the most significant obstacle considering that each State uses its proper language to record Patient Summaries and ePrescriptions inside its internal domain.

Nevertheless, eHDSI ensures interoperability acting as an intermediary capable of translating the healthcare information and also guaranteeing trust and security services during the medical data interchange. epSOS guarantees security only inside its framework, i.e. OpenNCP platform, that enables secure access to the clinical information of patients providing a standard data format to exchange them. Note that the epSOS project does not include data sharing, but only exchange. Furthermore, the eHDSI architecture must be scalable and flexible to avoid configuration problems if a new Country wants to join the programme, or if it is necessary to add new functions in a future implementation of the infrastructure mentioned above.

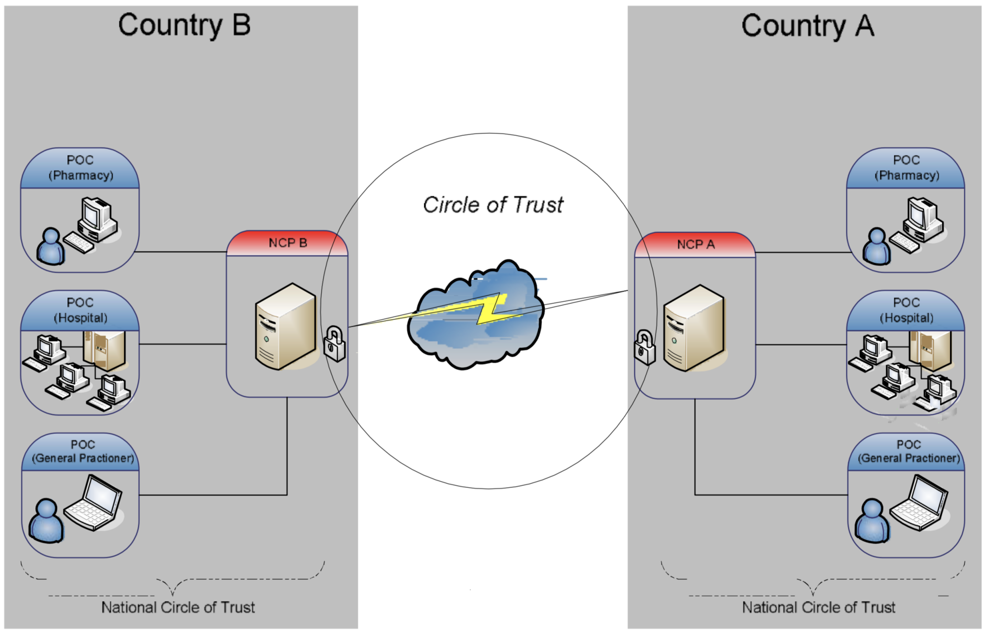


Figure 4.1. eHDSI Architecture: European and National eHealth domains (eHealth DSI System Architecture Specification).

The figure 4.1 shows a simple schema that represents how the eHDSI infrastructure interconnects Country A and Country B public domains. The domestic eHealth infrastructures, shown in grey, are composed of the already existing national health system, which remains unchanged, in addition to NCP node which is the only node that authorises each Country to interface with the opposite NCPs belonging to other Countries. The eHDSI architecture is characterised by a set of NCPs, which exchange medical data among each other. Note that every NCP interconnects domestic with community domain and vice-versa, so it acts as an inter-domain node, having one interface toward the Country's domain and another one toward the epSOS world: consequently every NCP belongs to both domains, performing different roles in each of them. In other terms, a single National Contact Point interacts with the healthcare area of a Participating Nation on one side, and with the epSOS area on the other using specific interfaces and for this reason, it is an essential part for both domains. At this point, inside the epSOS area, each NCP should directly communicate with other NCPs.

However, an individual NCP can successfully interact with their peers, if and only if a relationship of trust exists between them. Trust among Participating Nations is based on contracts and agreed policies and the trust relationships are organised in the Circle of Trust (CoT), id est, a closed network consisting only of NCP nodes that trust each other. In this way, if an NCP is part of this trusted circle, then it will be able to contact the others and exchange information in a security context: this security context is ensured by the eHDSI domain, of which the CoT is the concrete expression. In addition to this group of trusted NCPs, CoT also contains an identity provider who is responsible for storing and managing the identity information of peers that belong to it [37].

The figure 4.2 presents a high-level outline of what happens during an epSOS-OpenNCP transaction:

1. a foreign Patient (P) asks for medical treatment in a Healthcare Provider Organisation

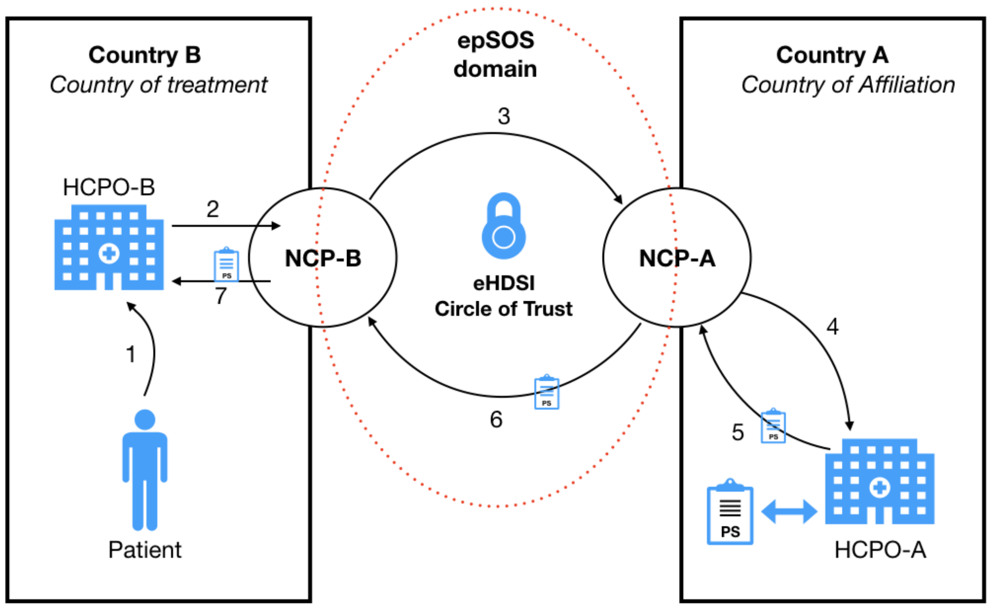


Figure 4.2. High level representation of cross-border interactions between two eHDSI Participating Nations.

(HCPO) belonging to Country B, the Country of Treatment;

2. the HCPO-B contacts his National Contact Point (NCP-B) to retrieve the patient's information from Country A, the Country of Affiliation, which is the Country of origin of the patient;
3. through eHDSI, NCP-B send a query to NCP of Country A (NCP-A) in which it requests the relevant medical record;
4. NCP-A checks if the patient exists in his domain and retrieves the clinical record from the internal healthcare system (HCPO-A);
5. then NCP-A replies to NCP-B, with a response containing the Patient Summary (PS);
6. NCP-B elaborates the Patient Summary coming from NCP-A
7. finally, NCP-B sends PS to healthcare structure in which the patient has demanded the treatment, the HCPO-B.

An overall epSOS activity involves the two Countries implicated in data exchange and the OpenNCP platform, which makes the exchange possible. The whole process is an intra-domain process, and it involves the following distinct realms:

- Country of Treatment. The area in which the foreign patient requests for medical treatments.
- Country of Affiliation. The place in which the patient holds his medical history because he has citizenship in this Country.
- eHealth DSI common area. An area that acts as an intermediary, intending to allow communication between the digital healthcare systems of European Countries, through the OpenNCP platform.

It is fundamental to mark that any area is entirely independent of the others. Country A and Country B domains, as well as the domains of other Member States, are absolutely autonomous because different entities manage them, i.e. Italy controls the Italian eHealth system, and they have no one possibilities to interact directly with each other; also the eHealth DSI is independent of all the Participating Nations infrastructures because the European Union handles it, and only the NCP gateway can put in communication the National area with the EU epSOS area. Although national and community spheres can interact each other through the National Contact Point, each entity is uniquely responsible for its environment from a technical, legal and security point of view, and every State is accountable for what happens in his infrastructure.

4.2.1 National Contact Point

European Member States can join epSOS through a particular gateway, the National Contact Point (NCP). NCP is the junction node that links each national system with the EU common infrastructure, working as an entry/exit point between the two domains. The OpenNCP Community provides a reference implementation of the NCP that includes only the epSOS side services; consequently, each Participating Nation has to develop its custom NCP, making it interoperable with its domestic infrastructure respecting a set of security rules and interoperability requirements established by the EU Commission. The fulfilment of this requirements is examined by a team of expert, i.e. Security Audit Group [38], who coordinates the security audit process and decides whether a Country satisfies the minimum conditions or not.

NCP node is the central element in the eHDSI infrastructure, and its functions are crucial for the Member State (MS) that implemented it, especially considering technical and legal points of view. In fact:

- Technically, NCP is the main module that links the eHDSI infrastructure with the MS internal infrastructure and acts as an interface between the two areas. All the cross-border communications of each Country must pass strictly through the NCP so that a Country cannot directly access to health services of another Country. The concrete implementation of NCP is under the responsibility of the Participating Nations, which has to extend the reference implementation to connect the NCP with their domestic systems, following the interoperability specifications. Furthermore, NCP handles semantics and technical adaptations, e.g. translation and transcoding of Patient Summaries, providing the medical documents in the language of the Country of treatment.
- Legally, NCP controls the flows of information to and from the MS using a legal support structure: it is the only legal agent of the MS that guarantees the compliance of its domestic health services with the eHealth DSI infrastructure.

An NCP is a “participant” in the eHDSI world if and only if it complies with the eHealth regulations in terms of structure, behaviour and security policies [39]. The internal logical structure of an NCP includes several logic blocks described below:

Data discovery exchange services allow the interchange of messages between national and EU domains, such as clinical information and patient identification data;

Transformation services include translation and transcoding services of the medical data from the National model to eHealth DSI format and vice-versa, in order to support the linguistic interoperability;

Support services manage the application functions necessary to guarantee availability, response time, delivery and session in the eHDSI infrastructure;

Trust services guarantee trust and validation, verification, signature, mapping of messages and data and also the patient consent mechanism;

Audit services collect together all the application services needed for traceability (e.g. logs) and auditing.

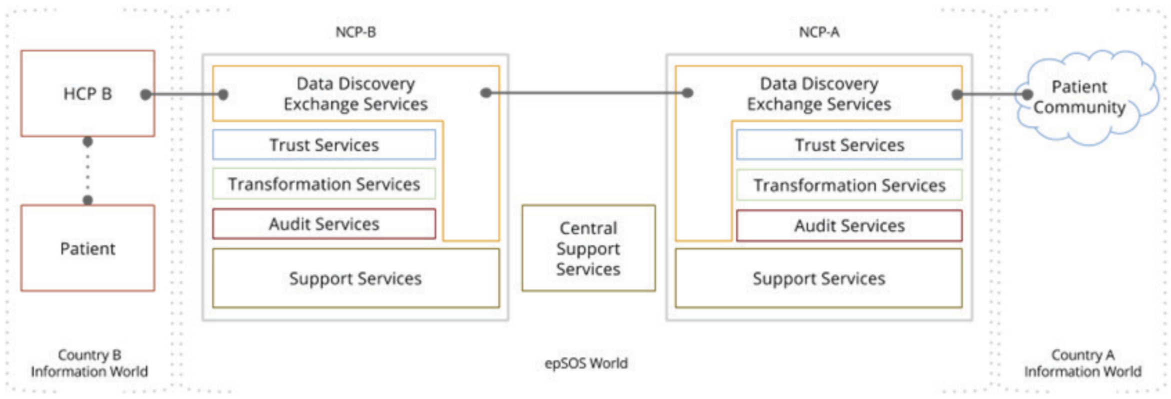


Figure 4.3. NCP high-level architecture diagram (eHDSI System Architecture Specification).

The reference documentation classifies these functions as internal and external: internal services concern the eHDSI side (e.g. communication between 2 NCPs), while external services deal with the interconnection between the national infrastructures and eHDSI.

Considering the Trust and Audit Services, they fulfil internal and external functions. As mentioned above, the NCP is a mediatory element that is part of both domains (National and European), and each entity is fully accountable for its operative area. Consequently, from a computer security point of view, it belongs to two security contexts that are distinct, separated and independent of each other; this separation is achieved through dedicated interfaces, one towards the Participating Nation and the other towards eHDSI. Therefore, any interface has its network security, its PKI and its semantics in terms of standard cryptographic algorithms and security policies: the EU Commission is responsible for the security context that involves epSOS/OpenNCP, i.e. the Circle of Trust, while the Member States are responsible for security that concerns the NCP interface toward his internal domain.

Assuming that the security level of the national context is adequate as is the European one, the NCP remains the most critical element of the whole system, precisely because it must manage the transitions between two security contexts that are administered by different entities.

4.2.2 OpenNCP Internal Components

NCP is a web service composed of several components, designed to achieve a Service-Oriented web application. The eHDSI services are classified in different profiles, among the others the:

eHDSI-XCPD Profile is responsible for discovering patient demographic information in his/her country of affiliation. Before the clinical data exchange, the patient must be identified in its origin country;

eHDSI-XCA Profile is responsible for processing patient information and data, i.e. the Patient Summary and ePrescriptions;

eHDSI-XDR Profile includes:

- an Order Service for retrieving the ePrescriptions;
- a Dispensation Service for registering any dispensation of drugs in Country-B, based on the requirements recovered from the Country-A;
- a Consent Service for managing patient's consent about the treatment of his sensitive data.

Inside the eHDSI domain, the structure of medical documents follows the Clinical Document Architecture (CDA) standard, i.e. an XML-based language defined to specify a standard for structure, semantics and encoding to exchange clinical documents. The language of the documents

Component	NCP (A/B)	Description
OpenNCP client connector	NCP-B	Allows portals (or other clients) to call the IHE clients. It exposes a non- standard SOAP interface.
XCPD/XCA/XDR client	NCP-B	IHE clients that call services exposed by other NCPs to implement the cross- border exchanges.
TRC-STS	NCP-B	Secure Token Service that issues an NCP-B-signed Treatment Relationship Confirmation SAML assertion.
Security manager	NCP-A/B	Implements PKI operations, e.g., digital signatures generation and validation.
Audit manager	NCP-A/B	Establishes communication towards the OpenATNA- based Audit Record Repository in order to assemble and persist audit messages. This repository can be consulted through a Web user interface.
eADC	NCP-A/B	Automatic Data Collector, persists IHE-transactions statistical data.
Assertion validator	NCP-A	Validates SAML assertions used in the IHE transactions.
XCPD/XCA/XDR server	NCP-A	Exposes services to other NCPs.
eHDSI server APIs	NCP-A	Set of interfaces that connect the eHDSI services to the national infrastructure through the National Connector.
Transformation modules (TM, TSAM, LTR)	NCP-A/B	Transformation Manager (TM), Terminology Services Access Manager (TSAM) and Local Terminology Repository (LTR) work together to perform the semantic transformations of clinical documents.
Configuration and maintenance (Terminology sync manager, Configuration manager)	NCPeH-A/B	Terminology sync manager (TSAM-Sync) synchronises the semantic assets from the Central Terminology Server (CTS) to the LTR. Configuration Manager performs CRUD operations on the NCPeH configuration properties database (some of these properties are fetched from the Central Configuration Server SMP).
Configuration and maintenance (OpenNCP- Gateway)	NCP-A	Web-based back-office which allows the creation, signature and publishing of the NCPeH configuration (metadata) in the Central Configuration Server (SMP).

Table 4.1. NCP components already developed in the reference implementation.

exchanged in the eHDSI domain is English. The Figure 4.1 shows the NCP internal components; among the others, the set of transformation modules is vital to ensure the interoperability of documents exchanged. The Transformation Manager component (TM) takes care about translating the medical data from eHDSI CDA syntax to the syntax of the national infrastructure and vice-versa, exploiting the services offered by the Terminology Access Manager (TAM): note that the translation concerns not only the structure of data but also the language in which data are written. Indeed, TAM supports TM in the translation of a clinical concept from the Country

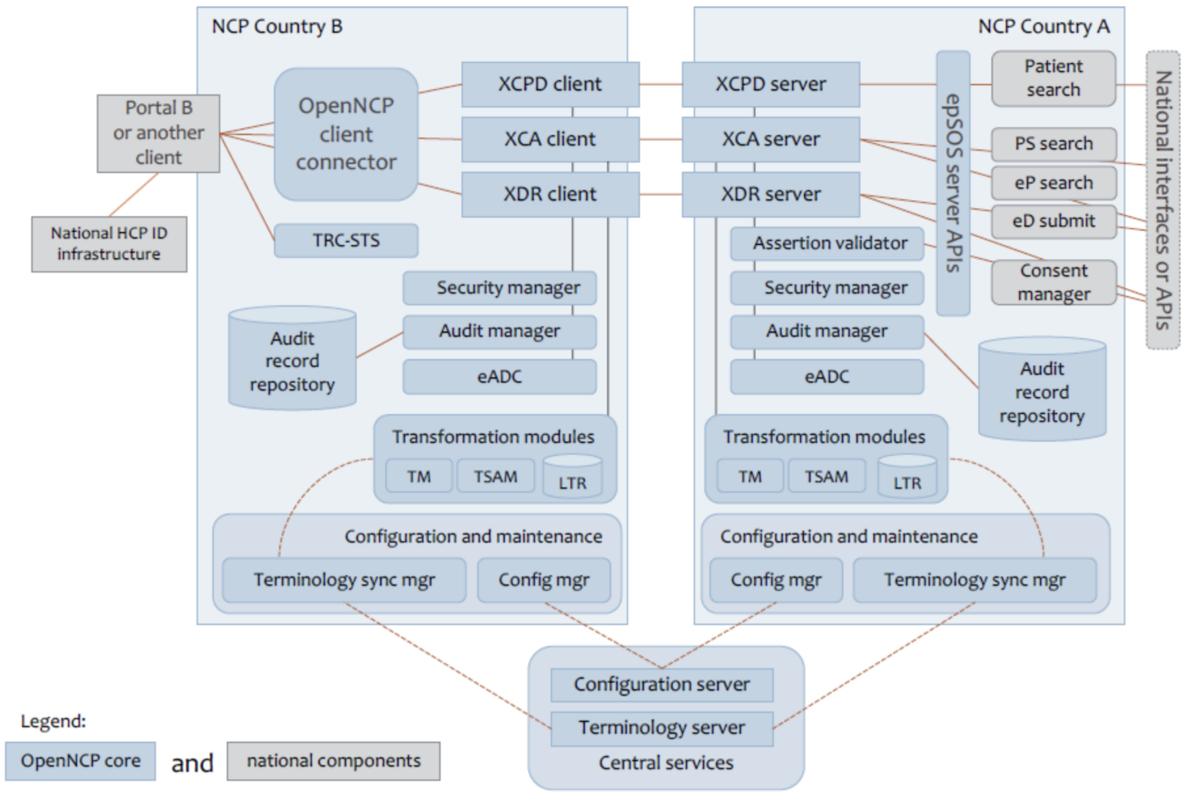


Figure 4.4. Internal components of the eHDSI National Contact Point. (eHDSI System Architecture Specification)

A language to English in NCP-A side, and then from English to the language of Country-B in NCP-B side. The Central Terminology Server handles the mapping activities, which are out of the scope of eHealth DSI and are under the responsibility of the National Linguistic Competence Centers from each country.

The table 4.1 explicates the OpenNCP core components (identified in blue in Figure 4.4), instead the national components are reported in the table 4.2 and marked in grey in the same figure.

4.2.3 HP and Patient authentication

The authentication of Healthcare Professional (HP) is performed at the national level, and it is out of the scope of the eHealth DSI. Indeed, the HP (i.e. doctor, nurse, pharmacist) can access to eHDSI system through the OpenNCP portal, developed and customisable by each Member State, after the authentication process in its national system. Therefore the eHDSI infrastructure and NCP-B are not involved in the HP authentication, but they trust of what done by the Country of Treatment (Country-B). Consequently, NCP-B accepts all the requests coming from Country-B. For what concerns the Patient authentication, the OpenNCP state of art leaves this task to the Healthcare Professional of Country-B: indeed, when an EU citizen requires medical treatment abroad, he has to interact with an HP in a medical structure. Before assisting the patient, the healthcare specialist authenticates himself in his national system and access to the OpenNCP portal, that establishes a communication with the NCP-B. At this point, the HP asks the patient to show him the European health insurance card, or as an alternatively, he can ask him where he comes from and what his Patient Identification number is. Note also that the HP inserts this kind of information by hand in a screen-page of the OpenNCP portal. After that, the NCP-B performs the XCPD patient discovery sending to NCP-A the Patient-ID of the sick citizen; thus, the NCP-A asks healthcare system of Country A if any person with the specified Patient ID exists in his domain. If a match exists, the patient discovery is gone right, and the HP-B can request

Component	**NCP (A/B)**	**Description**
Portal B or another client	-	End-user solutions and other middle-wares of country-B that provide cross-border eHealth features to health professionals.
National HCP ID infrastructure	-	Health professionals identity provider of country-B.
Patient search / PS search / eP search / eD submit / Consent manager	NCP-A	Implementations of the interfaces provided by the epSOS server APIs, that connect the NCPeH-A to the national infrastructure. Together they form the National Connector, which is a component developed by each country.
National interfaces or APIs	-	Interfaces used by the National Connector to call the different national services.
National patient ID/PS/eP/eD/consent infrastructure	NCP-A/B	National registries and/or repositories of patient identification data, patient consent and clinical data (PS and eP/eD).

Table 4.2. NCP components that each State must develop to ensure interoperability with his national healthcare system.

the Patient Summary in the next steps. Otherwise, the patient does not exist in Country A, and the OpenNCP platform cannot recover the patient information. As can be seen, the Country of Affiliation does not authenticate its own citizen through NCP, but it checks only if a Patient Code exists or not in his internal domain. The authentication of patient abroad is carried out by HP-B, which controls the European health insurance card of the citizen, and that is all. Therefore, OpenNCP needs of a cross-border authentication method for foreign patients, in which the States of origin of patients are also involved.

4.3 Patient Consent

Actually, in OpenNCP, the patient has to give consent for accessing clinical data in the country of residence (Country A). Therefore he does not do it directly: when the XCPD patient discovery has completed, in the HP portal appears a screen-page that ask for consent. The doctor can request consent to the patient, and possibly print a copy for him, but inside the system is the HP that acts to confirm the patient consent. Consequently, the role of Patient is passive because he has no one possibility to interact directly with the eHDSI system, and he needs the HP as an intermediary for his actions, which confirms the patient's decisions through the openNCP portal.

The following definition of "consent" is laid down in Art 4(11) of the EU General Data Protection Regulation 2016/679 (GDPR).

> The "consent of the data subject means any freely given, specific, informed and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her" [22, art. 4, comma 11].

In the health environment, the data subject is the patient, and the basic principles of patient consent can be summarised as follow:

1. The consent will be specific.

   Patient consent is acquired for the creation of "eHealth DSI patient data", i.e. Patient Summary. This action will take place in Country A, the Country of Affiliation. In addition, if a patient asks for medication abroad in Country B an additional consent is required, which concerns the access to patient data from abroad with a cross-border exchange.

2. The consent will be freely given.

   The patient has a free choice to participate in the eHealth DSI without any subsequent restrictions or negative influence to receive all necessary medical treatment or any other medical services. The patient may withdraw his consent at any time.

3. The consent will be informed. Consequently, the patient will be informed:

   (a) on the aims of the eHealth DSI, how his patient data will be used, on his rights and any other details of the processing of his data for the eHealth DSI purposes;

   (b) that his consent is free without any consequences if the consent will not be given;

   (c) that the collection and further processing of patient's health data solely for providing medical services, is a subject of legislation of a country in which medical care is provided.

   This information should be part of any binding agreement (e.g. Multilateral Agreement) signed by deploying Countries and be also part of the national standard information provided to patients for acquiring consent. In addition, the information will reside in the eHealth DSI NCP(A) and will, if required, be made available to the patient when he is in Country B [40, 3.3 Patient Consent].

4.4 OpenNCP Security Aspects

Since eHealth DSI deals with medical information, that is strictly personal and sensitive, OpenNCP platform must treat them with robust mechanisms that can ensure the fundamental security properties for the processed data, such as authenticity, integrity and confidentiality [41].

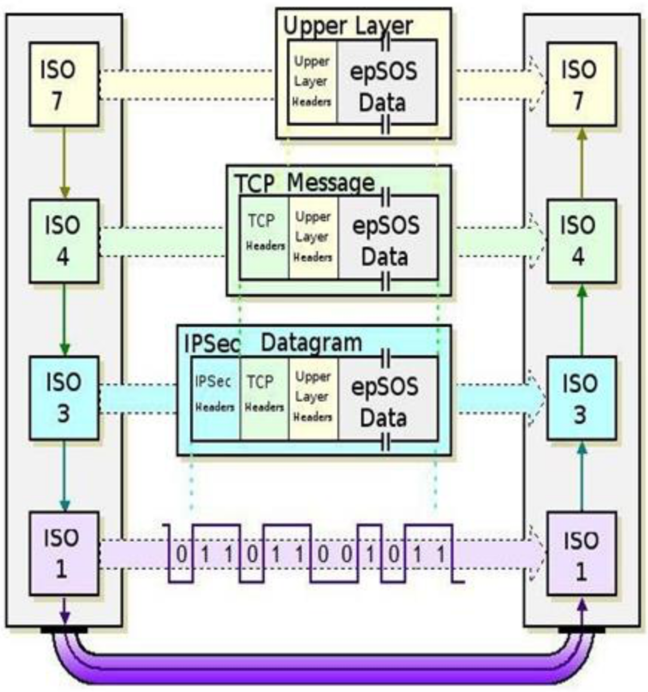


Figure 4.5. OpenNCP message exchange layers (eHDSI System Architecture Specification).

The platform employs cryptography and security protocols to ensure the security of communication of all the network layers. Inside the eHDSI domain, the transmission of data in the

Network Layer (Iso 3) is protected using a gateway-to-gateway VPN between all NCPs nodes. Transport layer Security (TLS) protocol is adopted to provide security of TCP messages, at the Transport layer (Iso 4), implementing end-to-end encryption between NCPs. Finally, at the Application layer (Iso 7) the format of messages exchanged in eHDSI is SOAP Envelopes with an XML payload in the SOAP body: all the eHealth DSI SOAP message must be compliant with the WS-I Secure Basic Profile [42], that defines a set of principles relating to the implementation of open standards for web services technology. All messages are signed with an x509 certificate.

The figure 4.5 summarises the kind of messages exchanged in each layer of the OSI model with the corresponding security protocol used to protect the eHDSI nodes. The end-to-end communication is between NCPs; hence, each NCP handles medical data in plaintext, making it vulnerable to a data breach.

Any participating NCP trusts of other NCPs in the eHDSI domain and the set of all participating NCPs constitutes the Circle of Trust (CoT). An identity provider gives the list of all trusted NCPs to each eHDSI node, which can establish mutual trust relationships between them. Note that each NCP also communicates with a national domain and the development of interfaces towards national areas is under the responsibility of the interested Country. Moreover, note also that eHDSI provides specific rules to satisfy security properties, but the project considers the protection against the propagation of cyberattacks out of scope. This decision, combined with the Circle of Trust, amplifies the critical aspects of NCP: consider that each NCP trusts its peers (CoT) and also trusts the domain of the State in which it is located. Consequently, if an NCP-A is admitted as a participant in epSOS world, other participant countries will accept this NCP, through their NCPs; therefore, attacks that compromised a National Infrastructure can exploit NCP to compromise other countries. In addition, if an attacker takes control of an NCP, he can have access to the entire infrastructure.

Chapter 5

HEALTHeID: eIDAS-OpenNCP Connector for eHealth

This chapter proposes a solution to solve the weakness of patient authentication in the epSOS environment: the solution proposal consists in the distribution of an eID Health connector capable of integrating eIDAS authentication within the openNCP platform, in order to obtain robustness and security in the authentication process during the identification of the patient abroad, exploiting his digital identity provided by his country of residence.

5.1 eIDAS cross-border Authentication for the OpenNCP platform

eIDAS introduces an electronic identification workflow, where the citizen requests a service from a Service Provider (SP), then the citizen is authenticated (through his national eIDAS infrastructure) towards the SP before the SP may provide access to its electronic services that the citizen is entitled. Regards to technical implementation, the Connecting Europe Facility (CEF) [35] provides a reference implementation following the eIDAS Technical Specifications v1.1, available under EUPL license, consisting of an eIDAS node components that can be used as a baseline to develop any country-specific eIDAS infrastructure, and allowing to connect different countries to the cross-border EU eIDAS node infrastructure.

Instead, the OpenNCP suite is a set of openly available and adaptable components which provides a reference implementation of the National Contact Point; NCP is the border gateway delegated by each country to enable the cross-border exchange of medical data, e.g. Patient Summary and ePrescription.

The HEALTHeID (HeID) connector is a software module which aims to improve the quality of health services provided to EU citizens by enabling their access to digital health records and facilities in Europe, integrating eIDAS to authenticate the patients and improving the authorisation process in the cross-border eHealth NCP transactions.

The figure 5.1, present a high-level overview of the resulting merged architecture, showing the role of the Health-eID connector and its task to integrate eIDAS within eHDSI OpenNCP. In this scenario, the HeID Connector interacts with the HP, the Patient, the eIDAS infrastructure and the openNCP platform:

1. the Patient (P) abroad requires for medical service in Country B, the Country of Treatment;
2. The Healthcare Professional (HP) asks to Patient where he comes from and email or phone number. This information is used by the HeID Connector to create an encounter between the Patient and the HP;

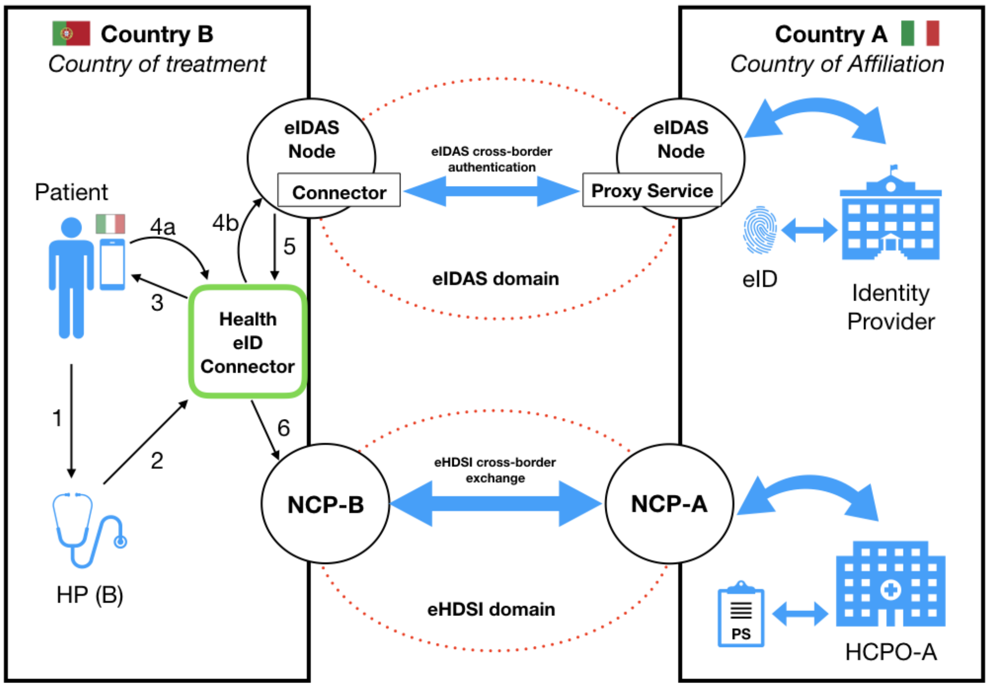


Figure 5.1. HeID Connector: high-level flow schema

3. HeID connector creates the encounter and sends a link to the Patient through email or SMS;
4. the Patient clicks on the link to accept the encounter and:
   (a) the HeID Connector confirms the encounter
   (b) the Patient is redirected to eIDAS for cross-border authentication, where he can authenticate himself in his country of residence with his digital identity;
5. HeID Connector receives from eIDAS the Patient attributes obtained from the authentication process;
6. Patient P is authenticated with his digital identity, and the openNCP workflow can start to obtain his Patient Summary or ePrescription.

Please note that this is a high-level overview; the following sections describe a fine-grained workflow that shows what happens in details and accurately reports the internal components of the HeID connector with their roles. Note also that Healthcare Professional is assumed already authenticated, since it is a national responsibility and depends on national infrastructure, in this case from country B, and is outside the scope of this project.

The HeID Connector implementation process must adopt definitions and concepts as described in the eIDAS Regulation and translate them appropriately to the cross-border eHealth context. Specifically, it is a component that allows interacting with the eIDAS infrastructure with the openNCP-based one: to merge eIDAS and eHDSI worlds, must take into account the requirements coming from both frameworks.

Those functional requirements are summarised as follow. Note that the keyword MUST express the mandatory status of the requirements while the keyword MAY expresses the optional one.

The HEALTHeID Connector:

- MUST adopt a coherent protocol profile to interact with the national eIDAS connector

- MUST have established a trust relationship with the national eIDAS connector.
- MUST provide an interface for the insertion of the patient identifier.
- MAY use the retrieved identification data to complete the patient identifier.
- MUST adopt authentication schemes coherent with the LoA used in the eIDAS cross-border authentication scheme.
- MUST provide an interface for the communication of the patient identifier towards the NCPeH component.
- MUST ensure lawful processing of personal data presenting to the user textual information about the foreseen use of the data, and the context of use (e.g. a specific healthcare encounter, a specific period).
- MUST implement an interface for the Patient to provide informed consent.
- MUST provide adequate input/output interfaces to allow patient use of personal devices (like a personal smartphone).

The principal point to solve is how to interconnect coherently the eIDAS authentication and the eHealth services provided through the NCPeH infrastructure. The link between the two worlds is the patient identifier: the cross-border exchange of information in the eHealth domain requires the existence of patient identifiers to select the correct information from national health information systems (through the XCPD Patient Discovery of openNCP). Some Member States use their national citizen ID as patient ID in the context (e.g. the Italian notified eID contains an attribute, the Fiscal Code - "Codice Fiscale"- suitable to be used as Patient Identifier), while some others use a sector-specific patient ID. The case can be even more complicated, since the patient ID, even if available at national level, may not be part of the data set retrieved after an eIDAS authentication exchange, due to the specific eID notified schema peculiarities (there may be many notified schemes per nation, and not all of them might have an attribute suitable as patient ID) and to the national rules on the matter (the Italian eIDAS node does not transmit the fiscal code in the response at the moment, even if at national level it is an issue under discussion).

OpenNCP state of the art foreseen an encounter between HP and Patient, the authentication of the Patient through the HP, and the provision of the patient ID from the Patient itself (as well as other relevant data to query the eHDSI infrastructure for patient data). As such, the patient ID is prone to errors and the authentication strength not always clear.

Thus, to provide a flexible solution suitable for all cases, the HEALTHeID connector must be able to differentiate on a national bases if any attribute is suitable for the purpose, in case yes, which one(s), and in case not, require it as an input from the Patient, coherently to the present cross-border eHealth scenarios. Since this procedure can be prone to error, it is necessary a verification step where the patient ID is sent to the country A eHealth infrastructure, and then verified versus local databases. The Patient according to privacy rights defined in the GDPR Regulation, has the freedom to give his/her consent in for the specific eHealth treatment, and this consent has to be adequately managed by the HEALTHeID connector.

5.2 Architectural scheme of the HeID connector for achieving an eIDAS/OpenNCP connection

The Service Provider in the figure 5.2 is the components where the HEALTHeID connector will reside and acts like the eIDAS SP toward eIDAS infrastructure and like OpenNCP Portal towards the eHDSI infrastructure.

This architecture presents an overview of the whole involved Service Provider components, but only some of them are in scope in the development of the HEALTHeID connector. The components inside the HeID blue square are the specific ones, the other (inside or outside the service provider) are component mostly already existing or depending on the national specific scenario. In particular, out of scope are:

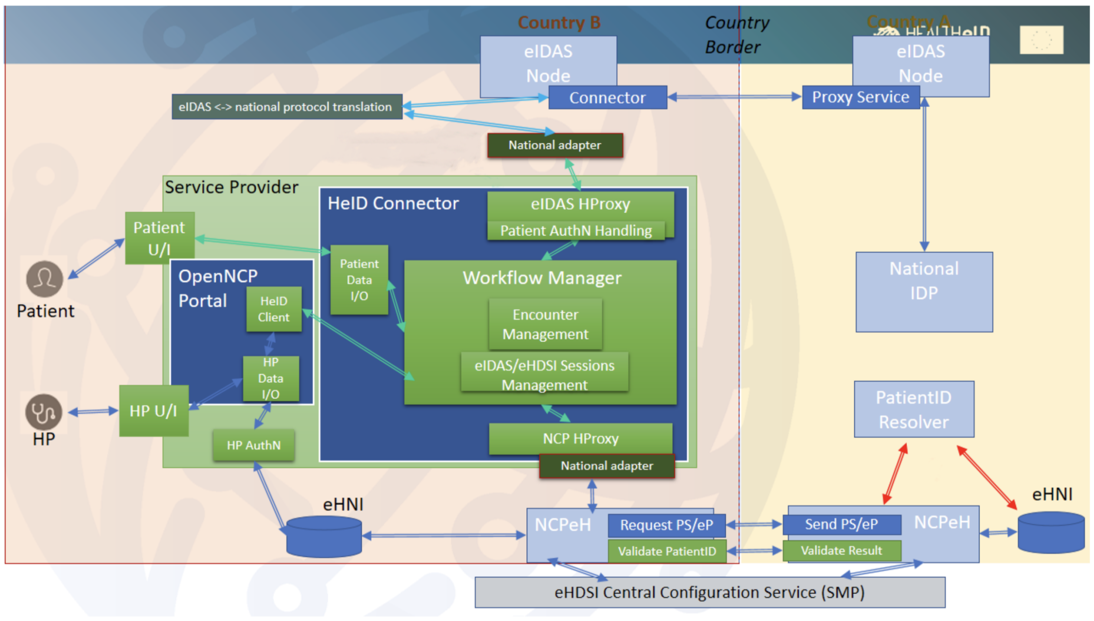


Figure 5.2. HeID Connector: internal components.

- Authentication Handling of the Healthcare Professional somehow performed according to local eH National specific scenario (through a local IdP)
- Creation of the encounter with the patient, since, as in the previous point, it depends on the national specific installation
- Expiry, by policy, of the HP session (this is again subject to the national specific rules and scenario)

On the contrary, in scope are the following functionalities:

- *Patient Authentication Handling*, done through eIDAS infrastructure. In this regard, the Workflow Manager is in charge to start the authentication signalling to "Patient I/O" module sending an anchor to the patient (e.g. SMS, email) in order to start the eIDAS authentication. The eIDAS authentication is started by the eIDAS HProxy component, which can get back the final result, after the eIDAS flow in place between the patient and the SP country, and communicate it to the workflow manager.
- *Encounter Management*, in particular, the workflow manager, which is 1) able to manage the Patient Authentication result and data; 2) able to propagate it to the eHDSI flow; 3) able to ask to the Patient further data not present in the data set provided by the eIDAS authentication (e.g. Patient Identifier in some countries, which has to be verified through NCPeH of the Patient origin country) and the Patient Consent to the treatment. In this sense, the workflow manager acts as an orchestrator of the HeID Connector business and realises the chain of operations calls.

5.2.1 Brief overview of the involved components

HeID Connector is structured as a set of sub-components to obtain a modular architecture which gives it properties of flexibility and scalability.

Table 5.1 shows the components involved in the Health-eID project; some components are an integral part of the HeID Connector, others are external but fundamental to drive the input/output interactions between all the actors. The Workflow Manager is an essential component because it orchestrates all those interactions and makes choices based on what happens in the entire system. Some components are more complicated than others, and they are described more accurately in the following sections. The not mentioned components, among those present in the Figure 5.2, are eIDAS or OpenNCP components and are out of the scope of this project.

Component	**Mandatory**	**Description**
HP Data I/O	No	This component is a part of the OpenNCP portal and manages the I/O operation performed by the Healthcare Professional (HP). Each Member State can create its own portal. This one is a reference implementation provided for demonstration purposes.
HeID Client	No	It is deployable within a portal, and his role concerns to make requests to the HeID Connector and receive responses from it. Each MS can create its client. The Health-eID project provides a reference implementation for demonstration purposes.
HeID Connector	Yes	This component provides integration between eIDAS and OpenNCP. It is composed by four sub-components: Patient Data I/O, Workflow Manager, eIDAS HProxy and NCP HProxy.
Patient Data I/O	Yes	Patient Data I/O consists of a group of HTML pages that allow the user to interact with the Workflow Manager. It also includes a Notification Adapter to send email/SMS to Patient; a custom one can replace this last one, that it is given only for demonstration purposes. The Health-eID project provides a default implementation with email features.
Workflow Manager	Yes	Fulcrum of the connector, it orchestrates all the other components managing the request and responses of Patient, HP, eIDAS and OpenNCP. It is responsible for the entire workflow.
eIDAS HProxy	Yes	Access point to the eIDAS world, it handles the eIDAS authentication process.
eIDAS National Adapter	No	The NationalAdapter module has been developed to fulfil the specific Italian needs, when an Italian patient has an encounter with a HP in another Member State. Anyway this module can be modified in order to be used in other scenarios.
NCP National Adapter	Yes	Allows the connection between the NCP HProxy of the HeID Connector and the NCP-B of OpenNCP
NCP HProxy	Yes	It works as a proxy toward the HeID connector and the OpenNCP world

Table 5.1. HeID components: the not mandatory components can be arranged by each Member States according to its preferences.

During the project activity and concurrently with the component's design, the following interfaces specify how to obtain the communication between components and which component has to interact with others.

Interface between eIDAS Connector and eIDAS Proxy Service

This interface belongs to eIDAS world and is up to the eIDAS technical specification; so it is out of the scope of this project.

Interface between HeID Connector and eIDAS / National translation component

This interface is subject to national specification, some countries, e.g. Italy, run or are running a specific Project to detail the involved protocols and profile, as well as to develop the related implementation. Indeed in Italy, which adopts SPID as National specification, FICEP project [43] developed the nodes and related translation, and, in particular, developed a component named SP Proxy between SPID SP and eIDAS Connector to translated eIDAS-SPID profile. This translation is fundamental in the Italian environment to integrate SPID with eIDAS and provide cross-border authentication of subjects. Therefore, considering the scenario in which an Italian healthcare structure installs the HeID Connector: if a foreign Patient asks for treatment in Italy, and wants to authenticate himself with eIDAS through the HeID, from a technical point of view, the HeID module performs an eIDAS authentication request toward the Italian eIDAS Connector. However, this request, on its way to reaching the eIDAS Connector, must be in SPID SAML format and, consequently, it is necessary to translate the eIDAS SAML coming from the HeID into the SPID SAML, in order to cross on the Italian SPID infrastructure; then the message must be re-translate from SPID to eIDAS SAML, having to enter into the eIDAS domain, as shown in the Figure ref:nationalAdapter. Some Countries reused the eIDAS protocol at the

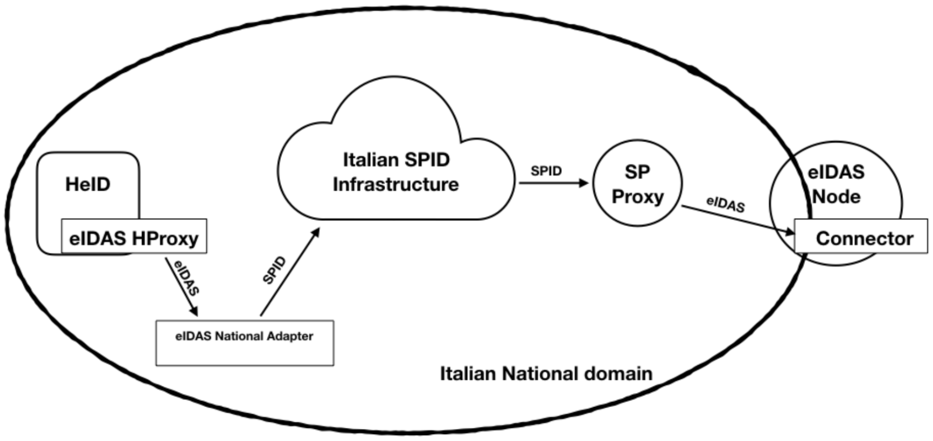


Figure 5.3. HeID and eIDAS National Adapter components in the Italian SPID infrastructure.

national level for their environment, so no translating component is required. Nevertheless, the eIDAS Adapter implementation depends strictly on the national infrastructure of each country, so it is out of the scope of this project.

Interface of the eIDAS HProxy component. In order to provide a common interface, re-use as much technical specification available and not lose any information where possible, the eIDAS Hproxy component adopt the eIDAS SAML profile for input/output communication.

Interface between eIDAS and National adapter. Since the eIDAS Hproxy component expects eIDAS SAML profile, countries adopting a different local protocol have the responsibility to develop a National adapter to translate from national protocol to the eIDAS profile. This adaptation process should have a minimum impact on the development side since a translator component, in such a country, should be already available and can be re-used for this task.

Interface between the NCP HProxy and NCPeH

This interface re-use the national specific implementation (e.g. SOAP-based) already developed, and thus under country-specific responsibility. The task of this project is to provide a stub for the NCP Hproxy component to allow each Member State to build up the component merging their national specific peculiarities. The communication between the NCP HProxy and the NCPeH-B is mediated by the NCP National Adapter (not to be confused with eIDAS National Adapter), which can act in two ways:

1. By directly calling the services exposed by the NCPeH-B. MS would provide an implementation where they would map the HeID message data to the data format of the services exposed by NCPeH-B. This method replicates the way the current OpenNCP Portal calls the NCPeH-B, through the OpenNCP Client Connector Consumer module;
2. By mapping the HeID message data to a national specific format. This plan is needed whenever the NCPeH-B services exposed to country-B national infrastructure are not the OpenNCP Client Connector Consumer ones, but national specific ones. This situation is the case, e.g., of Portugal, where the NCPeH-B exposes HL7 FHIR services towards the national infrastructure. In this example, the National Adaptor would map the HeID message format to HL7 FHIR messages expected by the Portuguese HL7 FHIR-based Portal Adapter (which in turn calls the OpenNCP Client Connector Consumer).

In the end, the National Adaptor implementation choice would be an MS responsibility, but in both cases, it is envisaged to be easily plugged into the NCP HProxy interfaces (e.g., following a similar approach to that of the NCPeH-A National Connector). The HeID project will provide an implementation of the first scenario.

Interface between the NCPeH A and NCPeH B

It is not changed in respect to OpenNCP specification, and out of the scope of this project.

Interface between the NCPeH A/B and eHDSI Central Configuration Services

It is not changed in respect to OpenNCP specification, and out of the scope of this project

5.3 Sequence diagram of the HeID connector workflow in an real scenario

This section describes the whole execution flow of a HeID session if a patient abroad wants to benefits of a medical treatment relying on the ePSOS OpenNCP services that are provided in a HeID scenario, which foresees an eIDAS cross-border pre-authentication given, by the patient before access to the OpenNCP platform. Note that the Healthcare Professional authentication is under the responsibility of his country, and it is out of the scope of the HEALTH-eID project. Before proceeding with the HeID workflow's description, it is necessary to consider the following assumptions:

1. The Patient owns an internet-enabled personal device, able to receive SMS (or configured to receive email)
2. HP and Patient authentication process is successful
3. The Patient is not a minor and is willing/able to provide consent
4. The Patient knows his patient ID
5. Country A can verify that the patient ID and eIDAS minimum data set match

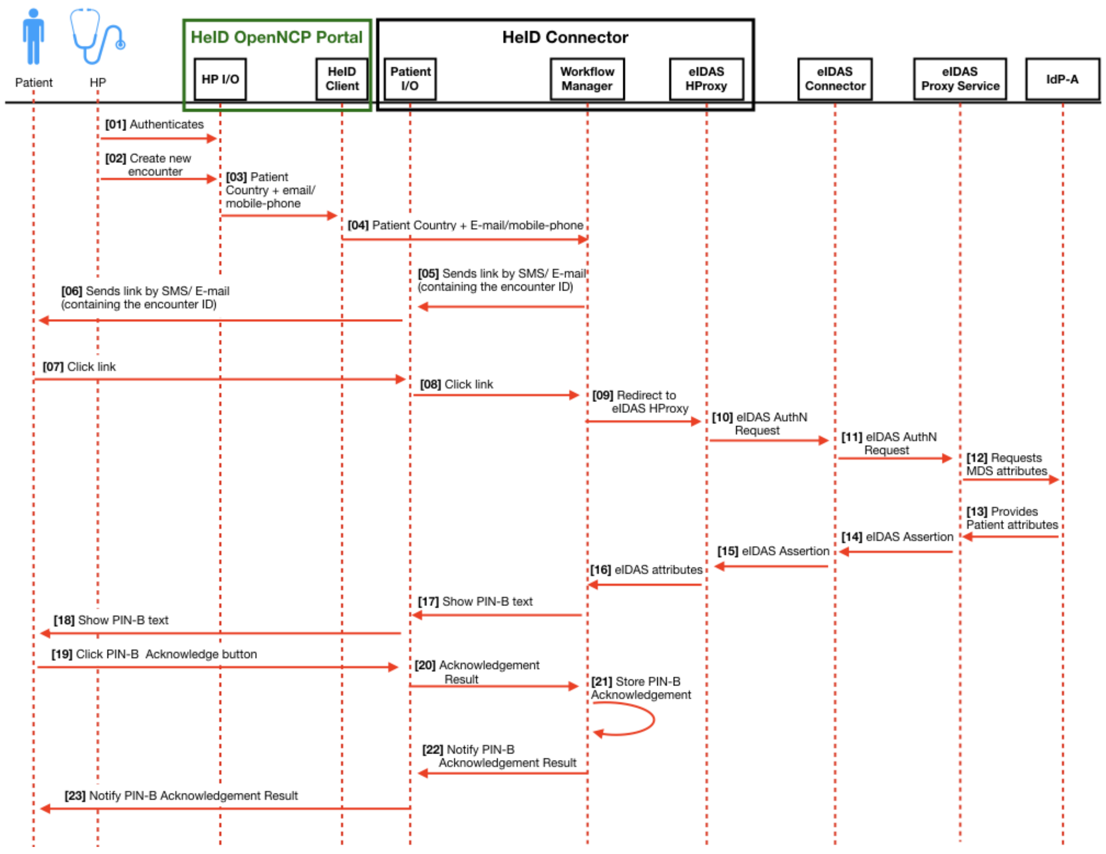

Patient
HP
HeID OpenNCP Portal
HP I/O
HeID Client
HeID Connector
Patient I/O
Workflow Manager
eIDAS HProxy
eIDAS Connector
eIDAS Proxy Service
IdP-A
[01] Authenticates
[02] Create new encounter
[03] Patient Country + email/ mobile-phone
[04] Patient Country + E-mail/mobile-phone
[05] Sends link by SMS/ E-mail (containing the encounter ID)
[06] Sends link by SMS/ E-mail (containing the encounter ID)
[07] Click link
[08] Click link
[09] Redirect to eIDAS HProxy
[10] eIDAS AuthN Request
[11] eIDAS AuthN Request
[12] Requests MDS attributes
[13] Provides Patient attributes
[14] eIDAS Assertion
[15] eIDAS Assertion
[16] eIDAS attributes
[17] Show PIN-B text
[18] Show PIN-B text
[19] Click PIN-B Acknowledge button
[20] Acknowledgement Result
[21] Store PIN-B Acknowledgement
[22] Notify PIN-B Acknowledgement Result
[23] Notify PIN-B Acknowledgement Result

Figure 5.4. HeID Connector: eIDAS authentication flow and patient consent.

6. The figures report all operations for a successful encounter, but the HEALTHeID Connector does not influence many of them. In the description, to mark this fact is used the `Out of scope (OoS)` abbreviation.

When a EU patient abroad ask for health service in another EU member state that implements the eHDSI infrastructure with eIDAS Authentication using the HeID connector, this is what happens:

(01)-(02) The HP authenticates himself in his Country infrastructure (OoS), accesses the Service Provider (Health-eID enhanced Portal of OpenNCP) via the web browser and chooses to use eIDAS authentication of patients. To apply the request, he has to create a new encounter with the Patient, interacting with the HP I/O interface provided by the HeID project and has to insert some patient information, such as the country of residence with the email address or the phone number.

(03)-(04) The openNCP portal elaborates the encounter request and contacts the HeID Connector to create the encounter.

(05)-(06) At this point, the Workflow Manager creates a unique encounterID that uniquely identifies the relationship between HP and Patient. Then the Patient I/O interface sends a link via email/SMS to the foreign citizen that asks for treatment. Note that the link contains the newly created encounterID.

(07)-(08)-(09) The Patient receives a notification with an encounter request, and he has to click on the received link to confirm the encounter. After this action, the Workflow Manager was informed by the "Patient I/O" and marks the encounter as accepted; then redirects the Patient towards the eIDAS HProxy.

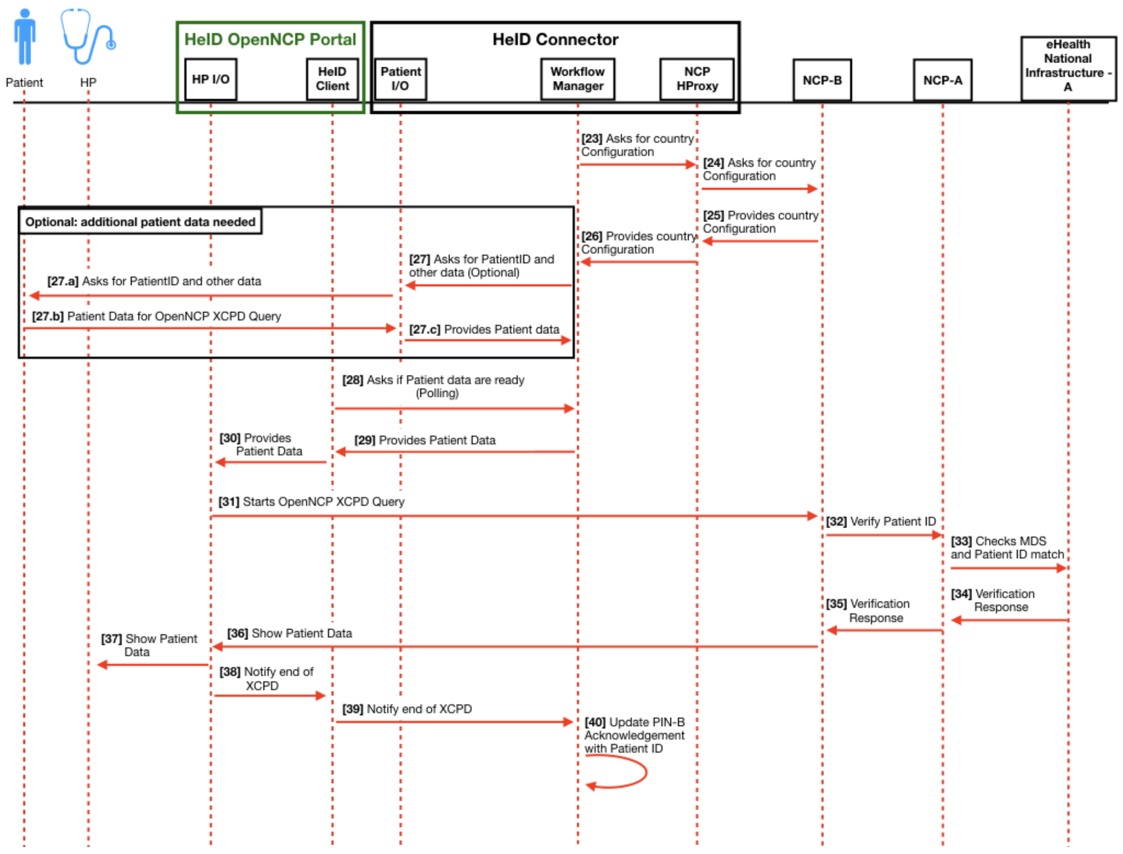

Patient
HP
HeID OpenNCP Portal
HP I/O
HeID Client
HeID Connector
Patient I/O
Workflow Manager
NCP HProxy
NCP-B
NCP-A
eHealth National Infrastructure - A
[23] Asks for country Configuration
[24] Asks for country Configuration
[25] Provides country Configuration
[26] Provides country Configuration
Optional: additional patient data needed
[27] Asks for PatientID and other data (Optional)
[27.a] Asks for PatientID and other data
[27.b] Patient Data for OpenNCP XCPD Query
[27.c] Provides Patient data
[28] Asks if Patient data are ready (Polling)
[29] Provides Patient Data
[30] Provides Patient Data
[31] Starts OpenNCP XCPD Query
[32] Verify Patient ID
[33] Checks MDS and Patient ID match
[34] Verification Response
[35] Verification Response
[36] Show Patient Data
[37] Show Patient Data
[38] Notify end of XCPD
[39] Notify end of XCPD
[40] Update PIN-B Acknowledgement with Patient ID

Figure 5.5. HeID Connector: OpenNCP XCPD Query (Cross-border Patient Discovery) flow.

(10) The eIDAS HProxy module prepares an eIDAS Authentication Request for the eIDAS Connector deployed in the country of treatment (Country B). Here starts the eIDAS authentication process.

(11)-(12) When the eIDAS authentication request reaches the Identity Provider of Country-A, i.e. the country of Affiliation, Patient is requested to provide to the IdP his/her credentials, and eIDAS authentication is performed using his/her device.

(13)-(14)-(15) During these steps, the eIDAS Assertion, that contains the Minimum Data-Set, is transmitted from country A to country B, and represent the patient authentication. The eIDAS HProxy computes the SAML assertion, checking its integrity and authenticity, and extracts the attribute values.

(16)-(17)-(18) Patient has correctly authenticated then the Workflow Manager stores his attributes; in the meanwhile, a Patient Information Notice is shown to the citizen. He must give his consent to take advantages from the eHDSI services.

(19)-(20)-(21) In the considered workflow, it is assumed that the Patient gives his consent. To communicate his decision, he has to click on the acknowledgement button; hence, the Workflow Manager is informed and stores the PIN of Country B with the Patient acknowledgement.

(22)-(23) This action is apprised to Patient, who receives a notification by email/SMS.

In this way, the initialisation of HeID session is completed with the authenticated Patient that has given his consent, after having seen the Patient Information Notice. The next steps concern the PatientID Discovery executed through the XCPD Query of eHDSI OpenNCP.

(23)-(24) Before to perform the XCPD Query, the system has to check if the Patient ID is present inside the eIDAS Minimum Data-Set. Then the Workflow Manager asks to NCP-B, through the NCP-HProxy, how is structured the eID schema of Country A and if the eIDAS Assertion contains the Patient ID as an attribute value. NCP-B contacts the eHDSI Central Configuration Service (this component is part of OpenNCP world and is omitted by the flow diagram) for specific country information, and if the Patient ID is available among the eIDAS data.

(25)-(26) NCP-B provides the country configuration to Workflow Manager, which can know if it already has the patient ID or if he needs to ask it explicitly to Patient. In the latter case, a new interface is shown to the citizen in which he has to insert by hand the patient ID and other data.

(27) Optional: Patient inserts by hand the missing data.

(28) During the last few steps, the HeID client polled the HeID connector asking for patient data, but since the data was not ready, the Connector could not send them. However, at (28), the Patient's data is ready, and the system has everything necessary to operate in the eHDSI domain.

(29)-(30)-(31) Then the HeID Connector sends to HeID Client the Patient's data (Patient ID included), and the HP can start with Patient Discovery though the OpenNCP portal, which performs an XCPD Query toward the NCP-B.

(32)-(33) NCP-A, located in the Country of Affiliation (i.e. the country of Patient), receives the Patient ID from NCP-B and checks if this unique identifier exists inside the National Infrastructure (NI-A). In this flow scenario, it is assumed that Patient ID is correct and exists.

(34)-(35)-(36) NI-A responds affirmatively to NCP-A; hence, NCP-A forwards the message to NCP-B, which informs the OpenNCP portal. Patient Discovery is completed with success.

(37)-(38)-(39) Patient data are shown to HP, and the HeID connector is informed about the correct Discovery of Patient.

(40) The Workflow Manager updates the PIN-B acknowledgement with the Patient ID provided by eIDAS (or by Patient) and confirmed by the eHDSI OpenNCP infrastructure.

Patient in country A exists, then the HP can process with the retrieving of Patient Summary:

(41) - (42) After updating the consent with Patient ID, HeID connector informs the HP, through OpenNCP portal, that the system is ready to retrieve the Patient Summary (PS) or an ePrescription (eP).

(43)-(44)-(45) HP submits the request in the "HP I/O" interface, the cross-border request starts in the NCP-B and reaches the NCP-A which elaborate the request and injects it inside the National Infrastructure of Country A (NI-A). It is assumed that the PS exist and the NI-A is able to find it.

(46)-(47)-(48) NI-A sends the requested PS to NCP-A, which translated it from NI-A format to eHDSI format; such translation also includes the language (country A language to English). NCP-B receives the PS, translate it in the language of country B and makes it compatible with the healthcare system of B, before sending it to Healthcare Professional.

(49)-(50)-(51) OpenNCP Portal shows to HP the medical document; all are ready to provide healthcare services to foreign Patient. A notification is sent to HeID connector informing it about the conclusion of cross-border exchange.

(52)-(53) Workflow Manager notifies the Patient about PS recovery.

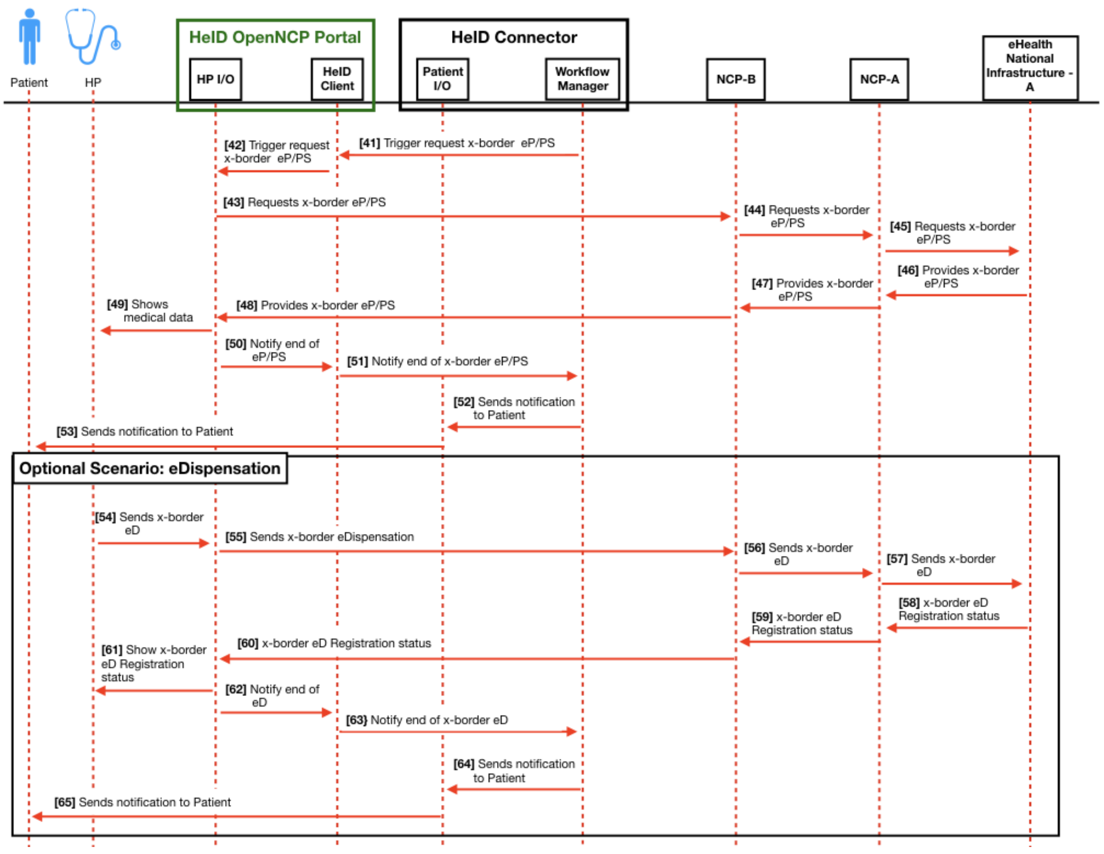

Patient
HP
HeID OpenNCP Portal
HP I/O
HeID Client
HeID Connector
Patient I/O
Workflow Manager
NCP-B
NCP-A
eHealth National Infrastructure - A
[41] Trigger request x-border eP/PS
[42] Trigger request x-border eP/PS
[43] Requests x-border eP/PS
[44] Requests x-border eP/PS
[45] Requests x-border eP/PS
[46] Provides x-border eP/PS
[47] Provides x-border eP/PS
[48] Provides x-border eP/PS
[49] Shows medical data
[50] Notify end of eP/PS
[51] Notify end of x-border eP/PS
[52] Sends notification to Patient
[53] Sends notification to Patient
Optional Scenario: eDispensation
[54] Sends x-border eD
[55] Sends x-border eDispensation
[56] Sends x-border eD
[57] Sends x-border eD
[58] x-border eD Registration status
[59] x-border eD Registration status
[60] x-border eD Registration status
[61] Show x-border eD Registration status
[62] Notify end of eD
[63] Notify end of x-border eD
[64] Sends notification to Patient
[65] Sends notification to Patient

Figure 5.6. HeID Connector: OpenNCP eP/PS exchange and eDispensation.

After the medical treatment, the doctor (HP) can record in Patient Summary the dispensation of pharmaceutical service. Note that the HP can not modify the PS directly, but he has to send an eDispensation notice to Country A with the new information to store in the patient report. The resumed process is shown in (54-64) and involves the two NCPs to exchange cross-border eDispensation. A notification is sent to the Patient after the eDispensation process in eHDSI (similar to other types of notification).

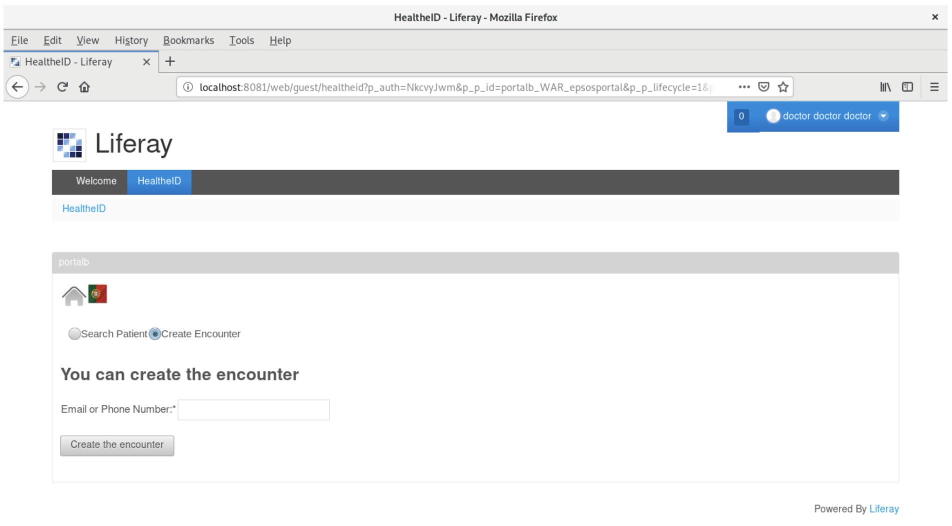
5.4 Description and design of the HeID connector components

5.4.1 Enhanced OpenNCP Portal

The `HP Data I/O` and `HeID Client` components are not part of the HeID Connector, but they are necessary to integrate the OpenNCP Portal with the HeID Connector.

HP Data I/O

“Healthcare Professional Data I/O” module activates an advanced version of the openNCP portal, which offers the portal all the functions needed to work with the HeID connector, in particular, it provides the Input/Output interfaces useful to the HP for interacting with HeID. This component is not part of the HeID Connector, but it is necessary for giving to the HP a communication interface with the system. Once the features are enabled, the country page in the OpenNCP Portal must display a radio-button to switch between the current eHDSI scenario (“Search Patient”), which performs the NCP Patient Discovery, or the HEALTHeID one (“Create Encounter”), which involves the eIDAS authentication, as depicted in Figure 5.7.


HealtheID - Liferay - Mozilla Firefox
File Edit View History Bookmarks Tools Help
HealtheID - Liferay
localhost:8081/web/guest/healtheid?p_auth=NkcvyJwm&p_p_id=portalb_WAR_epsosportal&p_p_lifecycle=1&
0
doctor doctor doctor
Liferay
Welcome
HealtheID
HealtheID
portalb
Search Patient
Create Encounter
Create the encounter
Powered By Liferay

You can create the encounter

Email or Phone Number:*

Figure 5.7. HEALTHeID-enhanced OpenNCP Portal.

HeID Client

Health-eID Client is a component that has the objective to ease the communication with the HeID Connector services, and it is intended to work not only in an OpenNCP Portal instance but in other Java environments. It performs the request from the HeID connector and receives the responses, so this is also not part of the HeID connector, but it is essential to communicate with it.

5.4.2 Workflow Manager

The Workflow Manager (WM) handles and orchestrates all the Patient and Healthcare Professional (HP) interactions during the Patient Authentication process, in order to provide a strong identification ensured by the eIDAS infrastructure. To perform a successful Patient authentication - we assume that Healthcare Professional is already authenticated -, the WM has to interact with four components:

- "HeID Client", which submits the HP requests towards WM.
- "Patient Data I/O", that provides an interface for the Patient in which he can submit decisions and receives messages from the connector.
- "eIDAS-HProxy", that generates the eIDAS SAML authentication request towards the eIDAS world, then elaborate the eIDAS SAML assertion and finally contacts the WM to communicate the eIDAS attributes.
- "NCP HProxy", it is a proxy component that contacts the NCP-B asking for information about the eID scheme of Country A, e.g. if the patient ID is included in the eIDAS attributes or if the system must ask the Patient to insert it by hand, as an additional attribute. Note that it the WM that contacts the NCP HProxy and not the contrary.

Through this components the WM handles the Patient and HP actions and integrates eIDAS with OpenNCP, managing the patient authentication and authorising the HP to retrieve the patient medical records. Its main functions are:

- create the encounter between Patient and HP,
- redirect the Patient toward eIDAS and evaluate the result of authentication,

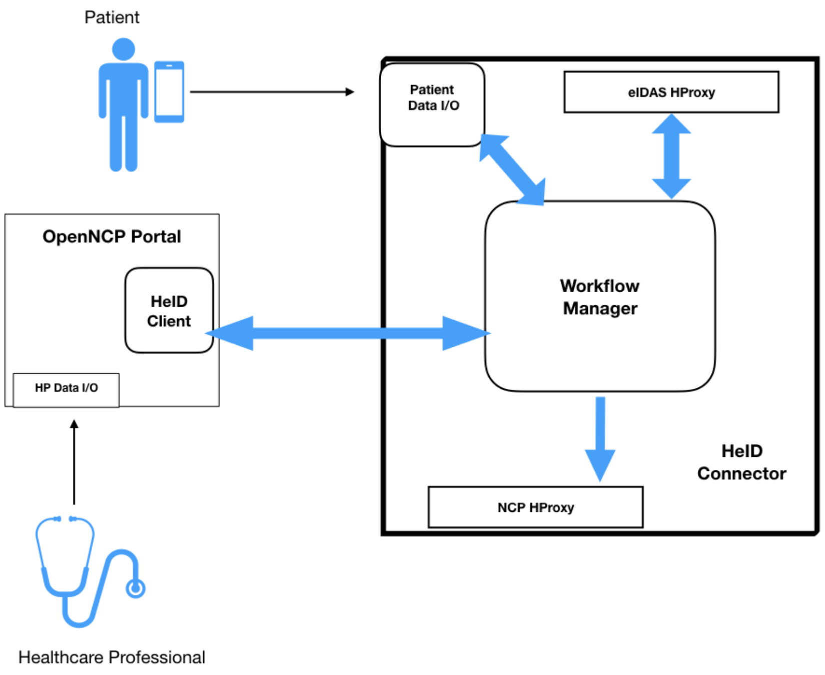


Figure 5.8. HeiD Connector: Workflow Manager interactions.

- ask and store the patient consent (given or not given),
- send notifications to patient about what is happening in OpenNCP (XCPD Patient Discovery, eP/PS process, eD transmission, PIN Acknowledgement/Consent decisions).

Encounter Management

The HeID connector receives the initial information sent by HP, that is the Patient's country and the Patient's email/SMS, then WM creates a new encounter which identifies univocally a relationship between the HP and a specific patient.

Such relationship is represented by a JSON Web Token (JWT), i.e. an open standard (RFC-7519 [44]) for safely exchanging information using a printable, compact and secure token, that exploits JSON objects, with the possibility to add a signature for verifying its authenticity. The token consists of three parts, the Header, the Payload and the Signature: the Header typically contains metadata information about the JWT, e.g. the algorithm used to sign the token, while the JSON data subjected of the exchange are inside the Payload. The last part, the Signature, is used to verify the integrity of the message. The authors consider JWT as a "compact and self-contained tool" because it is possible to represent all its parts as a compressed String in the format "*aaa.bbb.ccc*", using dots (.) as separator and where "*aaa*" represents the Header, "*bbb*" the Payload and "*ccc*" the Signature, all encoded in Base64. Un addition to the signature that ensures the integrity of data, it is also possible to use encryption to add confidentiality property: both signature and encryption are defined by other two standards, respectively the JSON Web Signatures (JWS, RFC-7515 [45]) and the JSON Web Encryption (JWE, RFC7516 [46]) and its a best practice to combine those standards together. With those characteristics, JWTs provide a lightweight solution to transport information realising stateless sessions, and for this reason they are suitable to characterise the Encounter in the HeID scenario. Indeed, the Workflow Manager generates the encounterID as a JWT token

encounterID = base64urlEncoding(header) + '.' + base64urlEncoding(payload) + '.' + base64urlEncoding(signature)

```
Example of JWT token base64 encoded (Encounter ID):

eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIxYzE0YTE5MC0wZmFjLTRlMTktYjk0Yi0yZ
   GViNzg3ODYwZjkiLCJleHAiOjE1NjMyMzYzMjEsInN1YiI6IklUIn0.foIhtBB2qH-
   vyKGOFcivYc54VeAVhg6GtPgaoEi68I
```

```
The same JWT token but base64 decoded:

   { "alg": "HS256" }

   {
      "jti":"1c14a190-0fac-4e19-b94b-2deb787860f9",
      "exp": 1563236321,
      "sub":"IT"
   }

   HMAC-SHA256(
      base64UrlEncode(header) + ." +
      base64UrlEncode(payload) ,
      sha-256 ( secret )
   )
```

Figure 5.9. Example of JWT's compactness with a base64 encoding. The colours are used to distinguish the different parts of the token: red colour identifies the header, green the payload and blue the signature.

in which the Header contains the algorithm used to sign the token (e.g. HMAC with SHA-256) while the Payload is a JSON object containing three properties:

- "**jti**": a JWT ID that univocally identifies the token, it is generated as a Universally Unique Identifier (UUID), i.e. an identifier standard knew also as Globally Unique Identifier (GUID) that guarantee the uniqueness of the ID [47]. This is the element that gives uniqueness to the encounter;
- "**exp**": a expiration time which identifies the expiration of the token and, by extension, the expiry of the encounter between Patient and HP;
- "**sub**": this property identifies the subject of the JWT and in practice, it contains the country-code of the Patient (e.g. "IT" for a Italian citizen that is requiring healthcare services abroad).

After creating the JWT Payload, the Workflow Manager computes the HMAC-SHA256(aaa.bbb, KEY) where "*aaa*" corresponds to the base64 encoded Header and "*bbb*" is the encoding of Payload in base64; instead the KEY value is the SHA256 of a secret, that is knew only by two entities in the HeID Connector, the Workflow Manager and the eIDAS-HProxy (that needs to read and validate the token for extracting the Patient Country, see **??**).

Figure 5.9 shows an example of Encounter ID generated by the Workflow Manager for an Italian patient abroad; note that the encounter ID is sent both to Patient and HP and creates a nameless contract between them that the Workflow Manager uses to link the requests of two actors. Patient receives the token as part of the link and when he clicks on it, he accepts the encounter and he will be redirect on eIDAS to authenticate himself in his country. Since the token does not contain sensitive data nor relevant information, as a design choice the JWT token is not encrypted but only signed by the WM, which verifies the integrity of each encounter ID that comes back re-computing the HMAC-256($aaa.bbb$, KEY) and comparing this value with the "*ccc*" part of the received encounter ID: if the two values are equals, the WM knows that the encounter is not tampered but generated by itself; then it can check the expiration time and if the JWT is not expired, the encounter is valid, otherwise it is not accepted.

Note also that the HeID Client, after the encounter creation, uses the JWT as authentication token in the HTTP Header: the WM checks and accepts this token if valid, but this behaviour makes the HeID Connector vulnerable to replay attacks (see **??**)

Patient Authentication and Consent

The electronic identification of the patient is needed to identify the person before providing a service to that person; in this scenario the service is the medical treatment supported by a cross-border exchange of sensitive data (Patient Summaries), such as data about the origin, genetic data and other information concerning health for which the processing is prohibited by the GDPR [22, art. 9, comma 2 (a)] without obtaining prior authorisation and specific consent of the involved person. The HEALTHeID project does not aim to provide a consent management service for a real scenario because it strictly depends on each country and its legal base; however a demonstration is provided to shows that the HeID Connector can be used in a scenario compatible with the last European Regulations which cares about privacy and data protection by offering a Patient Information Notice (PIN), i.e. an informative document shown to patient which concerns the purpose of the processing of his personal data, and a mechanism which stores the result of the consensus process. In particular, the CEF eHDSI Model PIN [48] is a template which aims to inform patients about their rights concerning the protection of their personal data and explain to them how those data are used by the system addressing the requirements explicated in the GDPR [22]:

- **Art. 13** Information to be provided where personal data are collected from the data subject,
- **Art. 14** Information to be provided where personal data have not been obtained from the data subject.

Each PIN Model is customised by each country, but it must follow the GDPR guidelines. The patient must provide the consent in Country A (the Country of Affiliation), when obtaining data, and in Country B when accessing data. Note that this context is affected by political decisions and some aspects are actually under discussion: the demo-consent service provided by HEALTHeID project is related to Country B, showing the PIN-B to the patient after the eIDAS authentication and before retrieving data from the OpenNCP environment. When the patient gives his consent, the OpenNCP XCPD Query starts, to check the existence of the PatientID in the OpenNCP side. Hereafter the result of the consent is stored in a DB with the following information:

- the `encounterID`, that is the JWT token in a String format,
- a `boolean` to mark the result of the acknowledgement,
- a `timestamp` of when the action took place,
- a reference to the `PIN` used for informing the patient on the purpose of the processing,
- the Patient Identifier (`Patient ID`).

If the consent is given, the OpenNCP data transfer can start. Otherwise, the patient can not take advantages from the eHealth services offered by OpenNCP.

5.4.3 eIDAS-HProxy

After the Patient clicks on the encounter link, he is redirected to eIDAS-HProxy, which is responsible for creating an eIDAS SAML authentication request towards the eIDAS world. This component receives the encounterID, from which it can read the citizen's country code stored inside the token: for getting the code, the eidas-hproxy needs to know the secret used to create the token, in order to validate it and check the signature (the "(*ccc*)" part). Hereafter, it builds the eIDAS authentication request storing the mapping between <Saml-ID, encounterID>in a `HashMap`, consequently, when the eIDAS assertion will come as a response, the component is able to identify the pair <Saml-ID,encounterID>and in turn retrieve the correct encounterID to attach at the received eIDAS attributes.

eIDAS National Adapter

The NationalAdapter module has been developed to fulfil the specific Italian needs when an Italian patient has an encounter with an HP in another Member State, but it can be modified in order to be used in distinct scenarios.

5.4.4 Patient Data I/O

This component is responsible for showing HTML pages to the patient and for collecting his inputs during an HeID session. It also includes a `Notification Adapter` which implements an email service for Patient notification. It receives the patient email from the Workflow Manager with the kind of notification to send, and sends an email based on the type of notification, e.g., communicates the encounter link to the patient or informs him about what is happening in the OpenNCP world (PS/eP/eD retrieval/submission, PIN acknowledgement or consent decisions).

Note that the `Notification Adapter` implementation is required, and each Member State can create its own, however, the Health-eID project provides a reference implementation with the email feature for demonstration purposes. To enable the sending SMS feature, the module needs of an SMS gateway (demonstration not provided).

5.4.5 NCP HProxy

The NCP HProxy component is responsible to provide the kind of configuration of the eHealth data structure in the country of Affiliation. This configuration is requested by the Workflow Manager after the Patient Authentication in order to map the eIDAS attributes with the OpenNCP attributes, checking also if some additional information are required because not provided by the electronic identification (e.g. the patient ID is not provided by eIDAS because this attribute is missing in the eID scheme of Country A).

NCP National Adapter

This National Adapter (to not confuse with the eIDAS National Adapter) allows the connection between the NCP HProxy of the HeID Connector and the NCPeH-B, requesting from the latter the country-A configuration (international search mask and eIDAS eHealth configuration).

5.4.6 NCPeH-A

NCP-A is now able to publish in the eHDSI Central Configuration Service (SMP) a new type of SMP file containing configurations related to the application of the eIDAS assertion data in eHealth: eIDAS eHealth Configuration. The OpenNCP-Gateway is the component responsible for the generation and publication of this file. For more information about NCP, see [33].

5.4.7 NCPeH-B

The NCP-B side was enhanced with a feature for fetching the eIDAS eHealth Configuration and returning it to the NCP HProxy via the National Adapter, which in turn uses the HEALTHeID-enhanced OpenNCP CC Web Services Client Consumer component, as depicted in Figure 3. This configuration is saved in the NCPeH-B file system, pretty much in the same way already done for the international search masks, with the filename such as EidasEhealthConfig-CC.xml, where CC is the ISO 3166-1 alpha-2 country code. This folder is created automatically by the component. For more information about NCP, see [33].

5.4.8 PatientID Resolver

This component is not provided by HEALTHeID, although its existence is assumed. It must be able unambiguously to resolve whatever data it receives from the eHDSI XCPD request into a patient ID that can be returned to country-B for the subsequent cross-border eHealth requests. However, the realisation is under the responsibility of the participating countries and outside this project.

5.5 Security considerations

5.5.1 Communication between components

The HEALTHeID Connector exposes some internal and external endpoints: internals is designed for the communication between the other software modules, while the external ones are aimed at the end-user. Those interactions are based on messages exchanged by the *HyperText Transfer Protocol Over SSL* (HTTPS), i.e. a variant of the HTTP protocol used for transferring web pages on the internet; but while in HTTP the communications are all in plaintext, the HTTPS version provides a safe communication creating a secure channel, encrypted through the exchanging of x509 certificates, which ensures the confidentiality of data and the identity of the involved parties.

5.5.2 Replay attacks

At the application level, the HeID Connector makes use of a signed JSON Web Token (JWT) for authorisation purposes. Specifically, the HeID Clients inserts the token on the HTTP Authorisation Header, in this way the HeID connector, before to process the request, reads this token and validates the requests if it is valid. However, this token is the same for each request, and it is never changed because it also corresponds to the encounterID, which uniquely identifies the business relationship between patient and healthcare professional. Consequently, the use of an unchanged authorisation token makes the HeID connector vulnerable to replay attacks.

A **replay attack** is a cyber attack in which the performer intercepts a valid data transmission and repeating it in the Internet network: specifically, an attacker can intercept the IP packets, usually belonging to an authorised user and intended for a Service Provider, copying and re-sending them toward the Service Provider: due to the validity of original data, the security protocols consider the attack as a legitimate transmission. Since the original data are only intercepted and re-transmitted, the attacker using replay does not necessarily have to decrypt them.

For avoiding this problem, it is advisable to use a **nonce** (i.e. a number, generally random or pseudo-random, that has a unique use) in the authentication header instead of using JWT token. Note that in our development, this token is used in authentication header to validates the request, as mentioned above, but it is also used to identify the encounter (because the token is the same and corresponds to the encounterID), and the HeID connector takes the encounter ID directly from the HTTP authorisation header. Therefore, for improving the current solution, this is what should happen:

1. the workflow manager (WM) creates the encounter, using a signed JWT token as encounterID (not changed).
2. The WM sends the encounterID to HeID client with a nonce that the client could be used in the next interaction.

3. Afterwards, the HeID client uses the nonce to perform the new request, inserting it in the HTTP authentication header, instead of the encounterID. However, the encounterID is useful information, so it must be present in the HTTP body,
4. The workflow manager (WM) receives the message and checks the nonce present in the authorisation header. If the nonce is valid, the WM can read the encounterID from the HTTP body and process the request. After that, WM generates a new nonce and sends it to the HeID client, inside the response; in this way, the client can use the new nonce in the next request.

However, in the current implementation, the attacker can not modify the data of the intercepted transmission without the HeID connector rejecting it, and also, considering that the JWT token expiration is about ten minutes, there is a limit to the effectiveness of the attack, which can occur in a time window dictated precisely by the expiration of the token.

Chapter 6

HeID Connector Implementation

The chapter describes some principal tool used during the HeID modules implementation. Note that the thesis work so as design and implementation activities were performed working with different EU partners; therefore, some tools as *Git* and *Slack* were used in order to simplify interactions, comments and exchange of ideas. In particular, among the other components, the candidate took care of implementing the `Workflow Manager` independently, the `Patient I/O` in collaboration with SPMS and the `eIDAS-HProxy` component co-operating with AUTh and ARIA (ex LISPA). Consequently, this chapter presents only the software modules directly realised and produced by him during the thesis work. For the same reason, only these mentioned components are included in the .zip archive that the reader can found attached to this document; for the full components of the HEALTH-eID project, please refer to the eHealth DSI OpenNCP repository, available on the website of the European Union[1].

6.0.1 Tools

This section describes the basic technological tools used for the implementation of logical components during the HEALTHeID project activity.

Java Programming Language

Java is an *object-oriented* programming language and processing platform developed by Sun Microsystems in 1995 [49]. On balance, the Java's strengths are its object-oriented characteristic and its portability: the former allow a better representation of entities inside the the programs and makes the code easier to maintain and extend [51]; the latter, the portability, is the most important characteristic which allows that Java code, once built, can be executed everywhere using a Java Virtual Machine (JVM) [50] independently from the operating system. Specifically, the JVM makes the programs not compiled in machine code (native) but in a sort of "intermediate" code, called `bytecode`, which is not intended for be executed directly by the hardware but which must, in turn, be interpreted by a second program, i.e., the virtual machine precisely. Another advantage deriving from the use of Java is the possibility of using different software libraries made available by the *Java Application Programming Interface* (API), a set of software libraries that provide support to the programmer, which can quickly invokes the prewritten code and performs high-level operations, reducing the realisation time and the probability error, and ensuring maximum execution speed.

Maven

Apache Maven is an Open-Source project management tool for the Java platform, which offers different services for simplifying some activities, such as library management, code compilation

[1]https://ec.europa.eu/cefdigital/code/projects/EHNCP/repos/health-eid/browse

and distribution [52]. The main reason to use Maven is to take advantages of the following services, provided by the tool:

- standardisation of the structure of a project and the compilation process,
- automated-testing and export,
- automatic management and download of the libraries needed for the project with a resolution of any transitive dependencies,
- ease of expanding the initial functionality by using plugins,
- automatic creation of a simple project documentation site.

The maven execution follows the information defined inside the *Project Object Model* (POM), an XML file which describes the general information of the project, dependencies, compilation process and: the `pom.xml` file clearly defines the identity and structure of a project, in all the aspects. An example of minimal pom is shown as follow,

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>eu.europa.ec.healtheid</groupId>
  <artifactId>healtheid-connector</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>war</packaging>
</project>
```

where:

- `<project>` is the root element of the XML file,
- `<modelVersion>` defines to which POM version this project complies,
- `<groupId>` is ID of the project group, to mark the context of the project or to specifies the organisation,
- `<artifactId>` is the ID of the component (or project, or module),
- `<version>` i.e. the project version,
- `<packaging>` which is the type of archive we want to export (e.g. jar, war).

The `groupId`, `artifactId`, `version` and `packaging` parameters will uniquely identify a project. If packaging is not specified in the POM, it will assume "jar" at default value. As mentioned above, one of the strengths of Maven is the management and the automatic download of the necessary libraries for the project: for exploiting this characteristic it is sufficient to add in the `pom.xml` file the dependency. For example, for including the OpenSAML library [53] in the project, it is necessary to add the following `<dependency>` tag:

```
<dependency>
        <groupId>org.opensaml</groupId>
        <artifactId>opensaml</artifactId
        <version>2.5.3</version>
</dependency>
```

Usually, at the first compilation, Maven connects to the central remote repository and downloads all the libraries necessary to compile the project. This starting process could require a little time; however, the subsequent compilations will take less time because Maven will store the libraries in the local repository, usually located at `/home/{$user}/.m2/repository` in a Linux environment, and no longer needs to contacts the remote one. The result of the compilation can be found inside the project directory, specifically in the `target` folder, including the final jar/war file to deploy on Tomcat.

Tomcat

Apache Tomcat [54] is one of the most popular open-source Web Container. Consider that a Java Web Application needs to an execution environment to be exposed as a Web server in order to make available his services for users or other components. In the thesis project, the version `8.5.35` of Tomcat is used for providing an isolated execution environment, on top of the JVM, for each component; components are deployed as Web Application in the *WAR* format (Web application ARchives), i.e. an archive which contains all the necessary data to execute the web application. These applications can be hosted inside a single Tomcat Container, in the form of archives characterised by the `.war` extension, and deployed in a dedicate environment, which is managed and controlled by the container itself: in this way the components "live" inside the container where they can be installed, configured and executed autonomously. For this reason, all the components developed in this thesis project are released in the form of WAR archives and can be deployed in the container discussed above.

MySQL

MySQL is an open-source relational database management system (RDBMS) [55]. His name derived by *Structured Query Language* (SQL), which is a standard programming language designed for interacting with the RDBMS databases in order to manage the data to store, e.g. inserting, deleting or modifying the data and interrogating the database system about information availability. SQL is one of the most widespread database languages in the world and MySQL, in essence, provides a database system able to store information following SQL syntax and instructions: the software is characterised by a server, in which the database is located, and by a client, which consents the user to interact with the server through the command-line. The database server can be used not only through his client interface, but also by various applications and web applications; for this purpose, a numerous API and connectors are provided in order to establish a connection between the MySQL Server and some clients applications, depending on the context. The power of these APIs is into providing low-level access to the database, exploiting their libraries and interfaces: for instance, since the Java programming language was used in the thesis environment, among the others [56] the *MySQL Connector/J* driver is used, specifically the `8.0.13` version, to enable a connection between the MySQL database server and the developed Java applications: this driver is written in Java and communicates directly with the database server through the MySQL protocol exploiting the *Java Database Connectivity* (JDBC) API [57].

Git and Bitbucket

Git [58] is a free and open-source distributed version control system, created to track progress and versions of the developed code during a software development activity. Practically, it is a command-line software that saves a series of "snapshots" of the filesystem in a custom specific location creating a sort of history of the temporal code version realised during the development process.

Bitbucket [59] is a free and secure *Git* that offers the possibility to manage multiple repositories and making them available online for free; generally, the repositories can be published online as public or private, by choice; the public repositories are typically used for share public project and open-source code that can be consulted, downloaded and inspected by anyone, instead, for the private ones, only the authorised users can access to the code and view it or download it. Apart from this, they can be easily shared and used by several programmers, each of whom can manage his own alternative versions (branches) and retrieve the code they need directly online. Besides, people can interact, review code, comment and take some discussions about the shared code with inline comments: those characteristics immensely facilitates the collaboration between remote people that are working on the same project.

Slack

Slack [60] is a platform for managing communication between workgroups, faster, practice and easily if compared to emails. This platform consents to create team chat, create thematic channels, share files or code fragments quickly and easily, working on them simultaneously, exchange direct and/or private messages with team members: overall, *Slack* simplifies business communications and was widely used during the thesis' work for communicating, designing and exchanging ideas with the other partners involved in the HEALTH-eID project, in order to consistently implement the various software modules and align their characteristics and behaviours, such as the choice of protocols, message format, kind of messages, choice of endpoints and other project business.

6.1 Technologies for the Development of Java Applications

6.1.1 Spring Framework

Spring [61] is an open-source framework born to simplify the development process during Java projects implementation: this simplicity is mainly due by the use of technologies such as the *Inversion of Control* (IoC) and the *Dependency Injection* that simplify the code development and allow the developer to focus on the essential application logic; in fact this tools can independently manage different technical aspects of software development, thanks to IoC, making the use of the framework very attractive. Also, the framework provides a highly configurable context for the creation and resolution of component dependencies, that are called *Beans*, which can be represented by an XML file or a Java class.

Inversion of Control: is an architectural principle that reverses the *Control Flow* of traditional programming. In the traditional environment, the developer has the responsibility to manage the programming logic, but, in Spring, the IoC makes sure that the programmer does not have to worry about things like creation, initialisation and invocation of object methods, and delegates these operations to the framework itself, inverting, indeed, the *Control Flow*.

Dependency Injection: it refers to a specific implementation of the IoC that deals with reversing the process of resolving dependencies, injecting them from the outside. With this pattern, it is sufficient to declare the dependencies an object needs and, when this will be instantiated, an *injector* will take care of solving the dependencies, using the IoC. The role of the injector is played by the IoC container, which takes care of instantiating the objects (*Beans*) declared in the project, finding and injecting all the dependencies associated with them. These dependencies can be components of the framework, or other beans declared in the application context.

Beans: a bean in Spring context is an object that is created, instantiated and managed by Spring; then this object is involved in the Dependency Injection mechanism. It can be configured through a Java class (defined with the "*@Bean*" annotation) or an XML document.

The Spring framework has a modular architecture and offers different features/functionalities, expressed by a single module: so then, despite being very large, it is possible to integrate within a project, by choice, only the modules which realised the interested functions and, however, those functions can be easily integrated into already existing projects. In particular, during the project activity, the following Spring modules were used: *Spring MVC*, *Spring Security*, *Spring Data* and *Spring Email*.

6.1.2 Spring MVC

Spring MVC is a specialisation of Spring based on the *Modern-View-Controller* (MVC) paradigm and specifically designed for the realisation of *loosely coupled* web applications [62]. *Loosely coupled* means that each component of the whole architecture can execute its tasks having no knowledge or

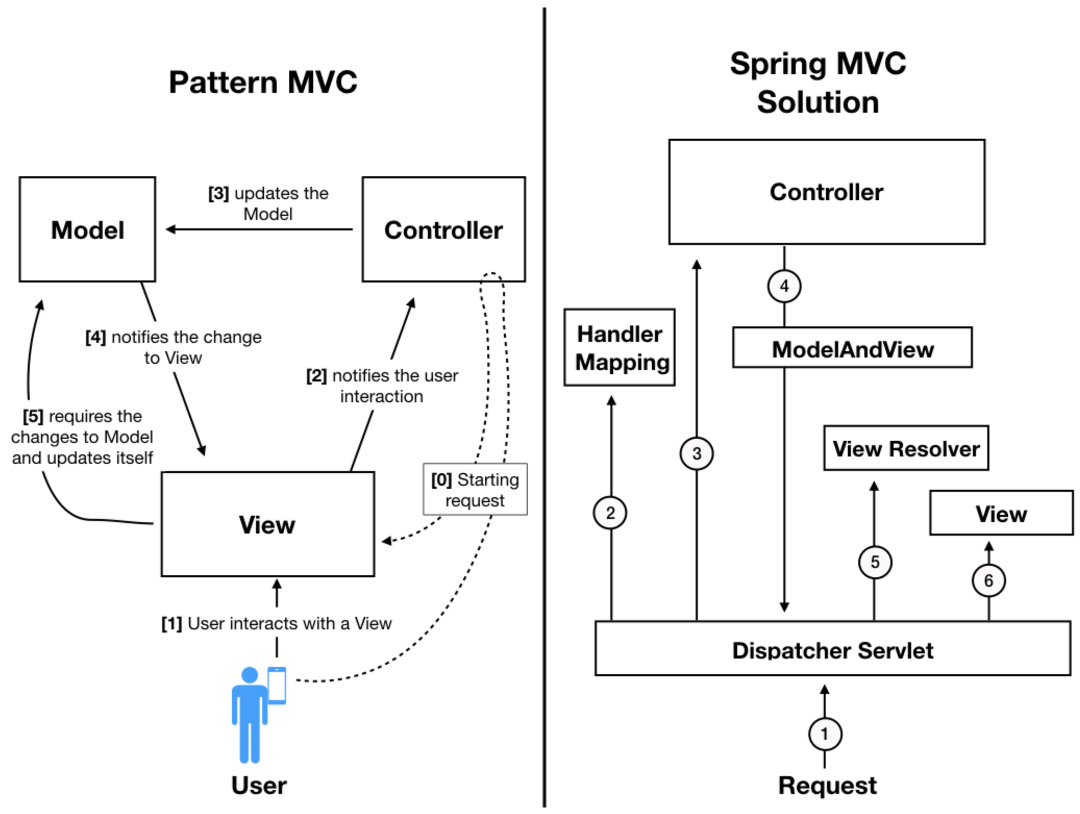


Figure 6.1. MCV paradigm architectural pattern, on the left, and the Spring MVC implementation, on the right.

a minimal knowledge about classes, methods and definitions of the others: this concept maximise the use of modularity and permits to replace one component with a new implementation without affecting or influencing the behaviour of the others. Besides, the MVC paradigm is based on the principle of a clear separation between *input logic*, *business logic* and *user interface logic* of a web application and realised this principle employing three elements:

- **Model:** it contains the methods to access or store data, such as objects and collections. A model describes and represents the data structures of the web application involved in the interactions with the user.
- **View:** it takes care of displaying the data to the user and manages the interaction between the latter and the internal infrastructure, showing the contents chosen by internal logic in a particular format (e.g. HTML pages).
- **Controller:** it receives the user's inputs from the *View* and triggers internal operations which usually lead a changing of state in the *View* and in the *Model*.

Spring implements the MVC pattern using an additional element, the *Dispatcher Servlet*, which is responsible for managing the MVC flow and for orchestrating all the interaction. Figure 6.1 compares the MVC paradigm with the Spring realisation: as displayed in the figure, in the Spring MVC side, the *Dispatcher Servlet* handles all the request, working as Front Controller, and in the next steps it drives all the MVC flow. Note that the *Handler Mapping* module is responsible to manage the mapping between the Java method and the URL invoked in the request, while the *ModelAndView* contains the data object and the name of the *View* associated with them and, finally, the *View Resolver* identifies the correct *View* to show at the user, based on the processing result of the request. In the thesis project, this specific framework interprets the role of the `Patient I/O`, which manages the interactions between Patient and HeID Connector.

Thymeleaf

Thymeleaf [64] is a template engine, a library written in Java. It allows a developer to define an HTML, XHTML or HTML5 page template and then fill it with data to generate the final page. Therefore it realizes a Model-View part of a Model-View-Controller model. The basic design principle of Thymeleaf requires that a model must be written correctly in (X) HTML.

6.1.3 Spring Security

Spring Security is a standard framework designed for giving security mechanism in the Spring-based applications [63]. It can be integrated with the Spring MVC slice and offers authentication and authorisation functionalities providing a configurable service for the access control. In the HeID Connector, Spring Security is used to filter the HTTP request, to configure the SSL certificates and enabling SSL connections.

6.1.4 Spring Data JPA

Java Persistence API [65] (JPA) is a framework that automatically maps the Java objects inside an existing relational database: create this manual mapping for the programmer is not trivial, because the Java objects and the relational database are two very different paradigms and translate the Java concepts in a relational entity or relationship could be very complicated. The JPA framework simplifies the programmer's work providing the so-called *Object-Relational Mapping* (ORM), which enables an integration service between the Java applications and the RDBMS systems. Via JPA, it is possible to map, store, retrieve and update Java objects from a relational database, using the correct annotations and interfaces.

Spring Data JPA [66] is a Spring module that integrates into Spring the JPA framework, simplifying the JPA working on the Spring-based applications further simplifying the work: it is necessary to declare the repository interface, and Spring will "magically" provide an automatic implementation. Explicitly, it consents to works with RDBMS databases using a coherent interface that relies on an abstraction called "Repository". Spring Data provides this `Repository`, which is a Java interface that can be `extended` from custom repositories for providing custom characteristics. In addition to the Repository interface, Spring Data also provides two other main interfaces: `CrudRepository`, which defines the contract for CRUD basic functions (creation, reading, updating and deletion); and `PagingAndSortingRepository` which extends CrudRepository by defining a contract for pagination and sorting. These three main interfaces ensure that application programmers can access data archives (such as relational databases or NoSQL databases) in a consistent way and with the possibility to easily change the storage without having to change the way the application interacts with the data store, therefore, if a user wants to change the database with another one, he can do it quickly and with no problem. In the thesis activity, Spring Data JPA is used to connect the Web Application with the MySQL database, in order to store the required information in the DB.

6.1.5 Spring Email

The Spring Framework implements a simple abstraction for forwarding email by employing the *JavaMailSender* interface [67]: it is necessary to have an email address with the credentials and the SMPT properties, as the listening port, to configure the module. The provided demonstration of the Notification Adapter exploits this service for sending emails to the patient, in order to communicate the encounter link and send the HeID notifications.

6.2 Technical details regarding the implementation of the HeID Connector module

The HeID Connector component is a Web Application implemented with Maven, for the project management, and *Spring-Boot*, for exploiting the different advantages that this framework provides in the software implementation: the "Spring Boot Maven Plugin" [68] is used for having a better integration between the two technologies.

For what concerns **Maven**, the `pom.xml` file is located in `$PATH/healtheid-connector/` folder, where $PATH is the filesystem location of the unzipped project's directory; the Project Object Model represents the configuration file which contains the details of the project dependencies, structure, and plugins. It is defined as follow,

```
<modelVersion> 4.0.0 </modelVersion>
<groupId> eu.europa.ec.healtheid </groupId>
<artifactId> healtheid-connector </artifactId>
<version> 1.0.0-SNAPSHOT </version>
<packaging> war </packaging>
```

with the `dependencies` which provides the Spring libraries for implements the Spring mechanisms and services previously described,

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-security</artifactId>
 </dependency>

 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-tomcat</artifactId>
     <scope>provided</scope>
 </dependency>

 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-jpa</artifactId>
 </dependency>
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-mail</artifactId>
 </dependency>
```

and the `dependency` for including the `MySQL Connector/J` driver, in order to obtain the JDBC APIs for communicating with the relational database.

```
 <dependency>
     <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
 </dependency>
```

About **Spring**, the framework organises the internal architecture following the *Model-View-Controller* MVC pattern. Spring exploits the *Java Annotations* to simplifies its use: annotations are meta-data associated with the source code, human-readable and similar to comments, but which survive after the compilation time. They can be associated in methods or objects without influencing the semantic; effectively, they influence the way the methods are treated, integrating them with tools and libraries. In summary, *Java Annotations* are indications that the programmer can specify to the `JVM`, defining how to treat the annotated objects during the execution phase. This mechanism gives the developer the possibility of using a simplified notation to make the code more readable and, in the meanwhile, underlining to the compiler the characteristics which a specific object must-have in the execution phase. In the following are defined the Spring annotation used in the implementation of the HeID connector:

- **@Controller:** This annotation is used to identify a class as a Controller, e.g. a web controller which handles HTTP messages and provides an HTTP access point to the application.
- **@RestController:** a specialised version of the Controller, which merges the *@Controller* and the *@ResponseBody* annotations. Indeed using this annotation, there is no longer a need to use the latter on all methods of the controller class.
- **@RequestMapping(String):** This annotation marks the HTTP handlers inside the *@Controller* class; in other words, it creates a mapping between web requests and the methods of the class, and it is used in the method and the URL to be mapped (the URL is specified as a String). To correctly use the annotation, it is a good practice to specify also the HTTP operation (e.g. GET, POST) as meta-annotation.
- **@GetMapping(String):** a specialisation of the @RequestMapping, dedicated for the handling of HTTP GET requests. The String value specifies the URL endpoint to map with the method.
- **@PostMapping(String):** a specialisation of the @RequestMapping, dedicated for the handling of HTTP POST requests. The String value specifies the URL endpoint to map with the method.
- **@RequestHeader(String):** This annotation indicates that a useful parameter should be present in the HTTP header. The String value specifies the kind of the HTTP header to inspect.
- **@RequestBody <T> object:** it indicates that an object of kind <T> should be present in the HTTP Body. This annotation must be used on the parameters of the methods declared in the controller class, and it automatically converts the HTTP body in the <T> Java object. Automatic validation can be applied.
- **@ResponseStatus:** it marks the type of the HTTP status code that the method should apply in the HTTP response.
- **@ResponseBody:** indicates that the returned Java object should be serialised in the HTTP body of the HTTP response. It performs the reverse operation specified by the @RequestBody.
- **@SessionAttributes(String):** it indicates the name of the attributes that should be transparently stored in the HTTP session.
- **@Component:** it indicates that the class is a special Bean ("component"), which is auto-detected by Spring and it is marked as belonging to the business logic.
- **@Autowired:** it marks a method or a field that should be managed by the Spring Dependency Injection mechanism. For instance, if a Java class contains another Java class, the declaration of the contained class is annotated, within the former, as @Autowired: in this way, Spring automatically takes care about initialisation and dependency management.

- **@Value ("conf.param"):** it marks a field to set with a `conf.param` specified in some configuration file. The use of this annotation consents to drive the Dependency Injection assigning custom values in an external file that will be injected inside the application by Spring.
- **@ControllerAdvice:** a specialisation of @Controller which declares an Exception Handler shared between all the component belonging to business logic. Therefore, using this annotation is possible to create a single class for handling the runtime exceptions and creating a specific method to manage the different exceptions.
- **@ExceptionHandler(Exception.class):** This annotation marks a method as the delegate handler for managing the specific "Exception" when the latter occurs.
- **@Configuration:** it marks this class as a class which contains one or more Bean methods that the Spring container may process.
- **@EnableTransactionManagement:** this annotation enables the transaction management of Spring Data, useful for configuring the communication between Spring and the relational database. In essence, this annotation enables a Spring interceptor for the transactional operation executed by the Java Persistence API.
- **@ComponentScan("eu.europa.ec.healtheid."):** this annotation consents to specified specific package for Spring scanning. If no specific package is defined, the scan starts from the package where the class with this annotation is declared. This annotation is useful if the programmer wants to organise his configuration classes in one package and the component to scan in another package.
- **@EnableJpaRepositories(basePackages = "eu.europa.ec.healtheid.repository"):** This annotation enables the JPA repositories. It is possible to specify the package of the repositories to avoid the Spring scanning of the whole project. Note that specifying the location of repositories consents also to use different databases for a set of repositories located in different packages.
- **@EnableWebSecurity:** This annotation includes the @Component and @Configuration meaning, and it is part of the Spring Security slice. Also, it marks the class as the container of the methods and Beans which provide the Web Security of the application (e.g. HTTP basic authentication).
- **@EnableGlobalMethodSecurity:** This annotation provides a set of others annotations to apply on methods for realising more granular security, e.g. in the access control it provides annotations to define pre-authorisation on methods, distinguishing the role of the user. In summary, it consents to manage security with more granularity handling each specific request.

6.2.1 Spring Controllers

The HeID connector interacts with different entities, depicted in Figure 6.2:

- the `HeiD Client`, which submits the Healthcare Professional requests. The `HeiD Client` component is external to the HeID Connector and is located inside the OpenNCP Portal.
- the Patient through the `PatientIO`, which reports his decisions and interactions. This component is represented by some *Views*, written with *Thymeleaf* and managed by Spring MVC, and by the `Notification Adapter` component that implements an email service for demonstration purpose.
- the `eIDAS-HProxy`, which generates the eIDAS authentication request towards the eIDAS Node-Connector deployed in the CoT (Country of Treatment) domain.

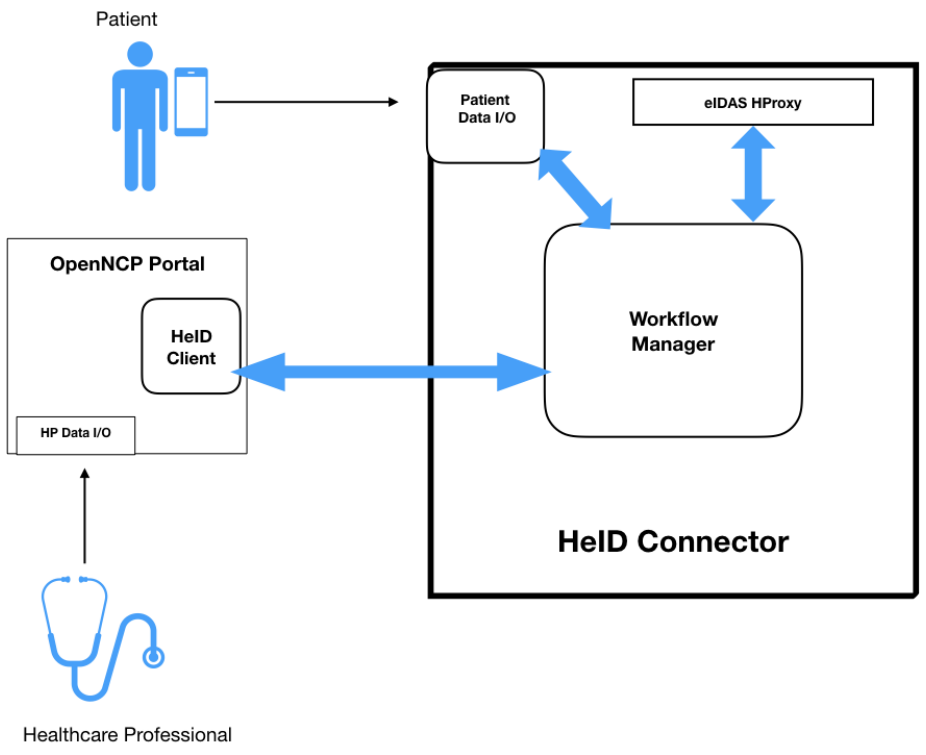


Figure 6.2. HeiD Connector and the internal bilateral communications. Note the absence of NCP HProxy, which is due to the unilateral interaction with the WM.

Taking into account that three entities are involved in the HeID Controller interactions, three different *Spring Controller* are defined inside the HeID connector project, one for each of them: `HeID-Client Controller`, `Patient Controller` and `eIDAS-Hproxy Controller`, respectively.

Note that the NCP HProxy component has no one dedicated controller even if it interacts with the Workflow Manager; this is why it does not perform requests toward the HeID Connector and, consequently, the HeID Connector does not need to expose dedicated endpoints for the NCP HProxy.

HeID-Client Controller

The HeID-Client Controller handles all the requests coming from the HeID Client component, which is driven by the Healthcare Professional (HP) actions. This controller provides three endpoints for invoking specific services:

- */encounter/createEncounter*: this endpoint triggers the business logic for creating an encounter between HP and patient. The encounter is represented by a JSON Web Token that will be sent to both, within the HTTP body to the former and via email/sms to the latter (inside the “acceptEncounter” link).
- */encounter/requestPatientData*: it is called by HeID-Client for checking if the eIDAS patient data are available in the WM. An HTTP message with `204 - NO CONTENT` will be sent as response if data are not yet ready (the patient has not completed the authentication process): in this case, the HeID-Client will have to try again later.
- */encounter/receiveNotice*: through this endpoint the HeID-Client can reports notifications about what is happening in OpenNCP, attaching the outcome of each process (`XCPD, eP-PS, eD`). The Workflow Manager is responsible to forward this notices to the patient.

Since the OpenNCP Portal, and consequently also the HeID Client, was designed to be *service-oriented*, it directly communicates via the HTTP protocol and does not need of *Views*. Therefore, the HeID Connector has to behave like a *RESTful* Web Service in the communication between the HeID Client and, for this purpose, the dedicated controller is implemented as a *REST* Controller in order to be able for managing URI and sending HTTP messages as responses, avoiding the use of Views.

Patient Controller

The Patient Controller is implemented in according with the MVC pattern and drives the `Patient I/O` *Views* shown to patient. On balance, the Patient Controller handles the requests though the following endpoints:

- */patientEncounter/acceptEncounter/*`{token}` : it is invoked when the patient clicks on the "acceptEncounter" link. The `token` is the Encounter ID generated by the Workflow Manager.
- */patientEncounter/additionalPatientData*: if the eIDAS authentication does not provide all the necessary data, the patient is redirected to this endpoint in order to trigger the right View and add the missing data by hand.
- */patientEncounter/patientAcknowledge*: this endpoint is invoked to call the View responsible for showing the Patient information Notice and for asking the acknowledgment to the patient.
- */patientEncounter/acknowledge*: it accepts the acknowledgment's result.

This endpoints are triggered by the actions that the user, i.e. the Patient, performs during the interaction with the Views.

eIDAS-HProxy Controller

The eIDAS-HProxy Controller receives the eidas attributes, through the *Model* and inside a Map Collection. This map contains the association between the *friendly name* of the attribute and the *value* of the attribute (see the eIDAS Minimum Dataset at Table 3.1). For receiving this information, this dedicated controller exposes the following endpoint:

- */heidconnector/acceptPatientAuthN*: used by the eIDAS-HProxy component for sending the eIDAS attributes to WM.

The different controllers are called by the Spring `Dispatcher Servlet`, which receives all the incoming requests and contacts the `Handler Mapping` for resolving the URL and mapping the request towards the correct controller.

6.2.2 Business Logic

Workflow Manager

The business logic of the HeID Connector is realised through the Workflow Manager (WM), which is the core component of all the infrastructure: it is marked with the "@Component" annotation and includes several instances that are annotated with "@Autowired" and injected by the Spring framework inside this class. The WM implementation contains all the necessary methods for satisfying the requests which will come from the Controllers, elaborating the responses, orchestrating the HeID workflow, and storing the necessary data on the configured database through the defined repositories. For what concern the repositories, the WM relies on the Spring Data JPA framework, which simplifies the programmer's work providing a standard `Repository` interface to extend: for instance, here is the Java code written inside the `EncounterRepository` declared to store the encounters that the HeID Connector will manage:

```
package eu.europa.ec.healtheid.repository;
import javax.transaction.Transactional;
import org.springframework.data.repository.CrudRepository;
import eu.europa.ec.healtheid.models.Encounter;

/ **
 * This interface defines an Encounter repository which stores the Encounter
     in the database
 * by its EncounterID. Note that this interface is not implemented and its
     working code
 * will be automatically generated from its signature by Spring Data JPA.
 * /

@Transactional
public interface EncounterRepository extends CrudRepository<Encounter,
    String>{

}
```

Note how it is sufficient to extend the `CrudRepository` provided by Spring Data, specifying the Java object to store as *Entity* in the DB and the Java property's type to use as *Primary Key*. The access to DB is granted by the methods extended from the `CrudRepository` interface.

Another fundamental thing done by the WM is the JWT token generation and validation: those actions are realised relying on the `io.jsonwebtoken` library which provides all the necessary methods removing all the management complexity for the programmer, giving a simple API to deals with the JSON Web Token. For using this library is sufficient to add the following dependencies in the Maven `pom.xml` file:

```
<dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt-api</artifactId>
</dependency>

<dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt-impl</artifactId>
       <scope>runtime</scope>
</dependency>

<dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt-jackson</artifactId>
       <scope>runtime</scope>
</dependency>
```

Exploiting the methods offered by this library, it possible to create and manage the mentioned token: in the implementation this tasks are delegated to a specific class `JwtTokenUtils`, which is autowired by Spring inside the Workflow Manager instance.

6.3 Technical details regarding the implementation of the eIDAS HProxy module

The eIDAS-HProxy component is responsible to create the eIDAS authentication request towards the eIDAS Connector of the Country of Treatment for allowing the cross-border authentication of the Patient abroad. This component is part of the HeID Connector, but it is implemented as a stand-alone Web Application, in order to provide a modular architecture. Especially, it was realised relying on the Spring framework and was designed exploiting the `eIDAS Sources (version 2.2.0)`, available in the eIDAS Node Integration Package [73]: the version `2.2.0` is chosen at the expense of the `1.4.3` because the `SAML Engine` of the former is based on `OpenSaml v3`, instead of the deprecated `OpenSaml v2` used in the latter. Consider that in the eIDAS environment, the *eidas-hproxy* component behaves as a Service Provider which requires a user authentication toward a remote Identity Provider, therefore, a design choice for the component realisation was to take inspiration from the eIDAS Demo SP source code, provided by eIDAS 1.4.3 and adapted to the SAMLEngine of eIDAS 2.2.0: this operation required a deeper analysis of the SAMLEngine 2.2.0 and, in particular, of its interfaces for the metadata generation.

In essence, the major differences between the SAMLEngine 1.4.3 and 2.2.0 are in the interfaces for the metadata generation: indeed the former version uses the `EntityDescriptor` interface provided by the OpenSaml v2 and located in the `org.opensaml.saml2.metadata package`, which includes all the metadata information.

On the contrary, the eIDAS SAMLEngine 2.2.0 reorganises the metadata generation using two interfaces, both located in `eu.eidas.auth.engine.metadata package`:

- **EidasMetadataRoleParametersI:** this interface represents either SP or IDP role descriptors, and creates an object accessible from EidasMetadataParametersI. It defines all the characteristics that the SP or IdP must have.
- **EidasMetadataParametersI:** this interface is responsible to present **all** data items for an eIDAS metadata, including technical and business information. It contains also the Role Descriptor data, defined by the EidasMetadataRoleParametersI interface.

To exploit the methods of the eIDAS SAMLEngine library, it is necessary to download the eIDAS libraries (more details in A.1.4) and add the following dependency inside the **pom.xml** file, located in the `$PATH/eidas-hproxy/`, where `$PATH` is the filesystem location of the unzipped project's directory:

```
<dependency>
       <groupId>eu.eidas</groupId>
       <artifactId>eidas-saml-engine</artifactId>
       <version>2.2.0</version>
       <scope>compile</scope>
</dependency>
```

Note that the *eidas-hproxy* must be able to read the JWT Token and, for this reason, it also needs of the maven dependencies related to the `io.jsonwebtoken` library; they are the same dependencies defined in the pom.xml of the *healtheid-connector* project and here reported for completeness:

```
<dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt-api</artifactId>
</dependency>

<dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt-impl</artifactId>
       <scope>runtime</scope>
</dependency>

<dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt-jackson</artifactId>
       <scope>runtime</scope>
</dependency>
```

About the communication with the other modules, the *eidas-hproxy* component exposes two endpoints:

- */eidas-hproxy/authentication*: through this endpoint, the Workflow Manager can trigger the building of eIDAS authentication request, sending the encounter ID as a parameter. Before sending the authentication request, the encounter ID is mapped with the SAML ID in order to link the corresponding response with the encounter.
- */eidas-hproxy/metadata*: this endpoint exposes a public interface, in which the eidas-hproxy publishes its SAML metadata; this interface is contacted by the eIDAS connector to retrieve the metadata necessary to start a SAML session. After the initialisation of the SAML session, the eidas-hproxy can send the authentication request to eIDAS connector.
- */eidas-hproxy/AuthResponse*: this endpoint is defined to receive the SAML assertion, which contains the eIDAS attributes of the authenticated user corresponding to the digital identity of the patient.

The eidas-hproxy component can be installed and configured by a configuration file, as described in the dedicated Appendix section A.1.4

6.4 Technical details regarding the implementation of the Patient I/O

The Patient I/O component is responsible for interacting with the patient for establishing the encounter, adding the data not provided by eIDAS and alerting the patient through HeID notification. It is characterised by two sub-components: the `Patient Views` and the `Notification Adapter`.

6.4.1 Patient Views

The Patient Views correspond to the implementation of Views defined by the MVC paradigm, which are used to realise the user interface: the source code of those Views serves only to build the HTML that has to be sent to the browser (data processing is managed, as mentioned above, by controller and model). They provide different web pages to show to the patient during the HeID workflow, summarised as follow:

- Acknowledge Page: through this user interface, accessible from the `/patientEncounter/patientAcknowledge` endpoint, the patient can accept the acknowledge. A dummy model is created for demonstration purpose, which points to the class-path file of the project (acknowledge.path=classpath:acknowledge/default.html) in order to be configured for using another customised file.
- Additional Data Page: through this page, accessible from the `/patientEncounter/additionalPatientData` endpoint, the patient can add additional information needed by HeID, but not provided by eIDAS.
- Error Page: This page will be shown if any internal error occurs.

The described web interfaces are developed using bootstrap4 [69], i.e. an open source toolkit for developing with HTML, CSS and JS) and are located in `$PATH/healtheid-connector/src/main/resources/templates`, where $PATH is the filesystem location of the unzipped project's directory.

6.4.2 Notification Adapter

The Notification Adapter is a subcomponent of the Patient I/O and is packaged as a deployable JAR file; the project directory is located in `$PATH/healtheid-notification-adapter`, where $PATH is the filesystem location of the unzipped project's directory.

This module is provided by the HeID project only for demonstration purposes and implements the email feature, that can be configured as shown in the dedicated Appendix section A.1.3. In the `pom.xml` file, located in `$PATH/healtheid-notification-adapter/` folder, the following dependency is added to exploit the Spring mail service:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

This dependency provides the library to use the `JavaMailSender` interface that consents to create and send a MIME message from a Java application: the *Multipurpose Internet Mail Extensions* (MIME) [70] is an internet standard which extends the RFC-822 and defines the email format to support text and multimedia content.

Chapter 7

Results

This chapter reports the results obtained from the integration of the HeID connector with the eIDAS infrastructure and the notifications sent by the Notification Adapter to the email address of the patient involved. The business logic of the Workflow Manager was tested with JUnit tests. Additional tests about the healthcare professional interactions and the integration with the OpenNCP platform were performed by the SPMS and CMS partners, principally because they are directly involved in the implementation of those aspects.

In particular, the testing of the eidas-hproxy module, which is responsible for creating and forwarding an eIDAS authentication request, was carried out within the same virtual environment employed for the development, using a docker eIDAS infrastructure, provided by the *TORSEC* group of Politecnico Di Torino, completed with all the components necessary for simulating of a real eIDAS workflow and an Identity Provider configured for containing some dummies digital identities used for testing purposes. In detail, it was found that:

- metadata are correctly generated by the eidas-hproxy component at the exposed URL and they correctly retrieved by the eIDAS node connector deployed in the docker environment.
- the eIDAS SAML request is correctly generated and accepted by the eIDAS connector; then, the eIDAS authentication flow continues without errors.
- the eidas-hproxy receives correctly the eIDAS assertion and elaborates the patient digital identity.
- the Workflow Manager is able to receive and elaborate the attributes provided by the eidas-hproxy.

Concerning the Notification Adapter, which is responsible for sending some business information to the patient, it is possible to state that:

- the patient receives correctly the encounter link at the indicated email address and he is correctly redirected to eIDAS by clicking on it.
- the HeID notification are successfully received in the email address indicated as patient email.

Chapter 8

Conclusion

The thesis' work described in this document is the result of active participation at the HEALTHeID project: which wants to present a technological solution for combining the eIDAS digital identity system with the OpenNCP eHealth data exchange system, in order to give in the latter a secure authentication of a patient abroad, who wants to benefit of a better health service by taking advantage of the epSOS infrastructure, which retrieves his clinical information from his country of residence in a safe and reliable manner.

The proposed solution consists of the deployment of a new module that has to be installed within the National healthcare infrastructure, the HeID Connector: this component is able to create a connection for exploiting the services offered by eIDAS and OpenNCP infrastructures, managing the whole workflow and providing an active role for the patient, who uses his smartphone to interact with the system during the authentication process and to give his consent. Also, he receives notification about the status of the process. This HeID solution for patient authentication is developed taking into account that the module has to be installed inside a national infrastructure, therefore, it is thought to be highly configurable by each Member State according to its internal system and preferences. The HeID connector can be improved in a future implementation:

- changing the HTTP authorisation mechanism, which employs in any HTTP message the same token, the encounterID, as authorisation token: this choice makes the system vulnerable to replay attacks. For avoiding this problem, it is advisable to use a *nonce* (a number, generally random or pseudo-random, that has a single use) in the authentication header instead of the JWT token,
- signing and validating the token with x509 certificates, instead of using a secret shared between the *eidas-hproxy* and *heid-connector* modules, as provided in the demonstration. This improvement can be efficiently made relying on the JSON Web Token library already employed to manage the token.
- verifying and updating the consent management, which is provided only for demonstration purpose, because it is strongly affected by regulations and political decisions that are actually under discussion.

In summary, the HeID solution integrates into OpenNCP the eIDAS authentication service: in this way, the user/patient can apply, in another State of the European Union, his digital identity, issued by his Country of Origin, for taking advantages of the eHealth services offered by OpenNCP, which actually does not offer a reliable patient authentication mechanism.

Appendix A

User Manual

This appendix describes the several components developed in the HEALTHeID project and gives a reference manual for installing the modules, explaining their configurations with a description of the actions to take and the properties to set in the configuration file. Here are described the configuration of modules implemented by the candidate during the thesis work and not contains the configuration of components realised by the other partners. The source code of all the components provided by the HEALTHeID project is available at CEF Digital OpenNCP Bitbucket [71]:

A.1 HEALTH-eID Configuration

The following sections give all the useful information for a user that wants to install the HeID Connector, developed in the thesis project, providing the set of instructions for configuring system environment and HeID components.

A.1.1 Software Requirements

Linux or Unix Operating System. The development, execution and testing of the thesis' work took place within the Ubuntu 18.04.3 LTS *bionic* distribution, installed on a Virtual Machine, and based on *Kernel* `4.15.0-65-generic`. As a rule, it is suggested to use the same Linux distribution to install the components developed during this thesis' activity. During the configuration, the user has to define some environment variables, which are useful for creating some pointers to specific paths and memory addresses[1].

Oracle Java SE Development Kit 8. The Java Development Kit or JDK is a set of tools for the development and execution of software written in Java language. Usually, the JDK contains within it a Java Virtual Machine (JVM), the Java Runtime Environment (JRE) and a series of tools and libraries for development. This is an essential and fundamental component needed to execute the developed application within the working environment: in particular, during the project activity, the `JDK 1.8.0` was used, where `1.8.0` refers to the installed version. This Java JDK was installed following the installation guide for Linux platform, provided by Oracle at [72]. The path of the JDK is `/usr/lib/jvm/java-8-oracle/jre` and an environment variable `$JAVA_HOME` is defined for pointing to it. It is possible to verify the correctness of the JDK tooling typing the following command in the Linux Shell:

```
$ java -version
```

Here is reported the expected output:

[1]How to create environment variables on Ubuntu - https://help.ubuntu.com/community/EnvironmentVariables

```
java version "1.8.0_191"
Java(TM) SE Runtime Environment (build 1.8.0_191-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Maven. The Apache Maven project management tool is installed by downloading the archive containing the software's binary files and its configuration in the operating system. Note that this specific version can be retrieved from the Maven Web site[2], and installed by downloading of the compressed archive `apache-maven-3.6.0-bin.tar.gz` which can be extracted in the `/usr/share/maven` directory. It is possible to verify the correctness of the maven distribution typing the following command in the Linux Shell:

```
$ mvn -v
```

Here is reported the expected output:

```
Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 1.8.0_191, vendor: Oracle Corporation, runtime:
    /usr/lib/jvm/java-8-oracle/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-65-generic", arch: "amd64", family:
    "unix"
```

As the user can see, this command allows him to check the installed version of the Maven software and the JDK development kit, in addition to the installation paths for both.

Apache Tomcat. The Apache Tomcat application server is installed in the version `8.5.35`, with a procedure very similar to Maven, reported above. This specific version of Tomcat can be retrieved from the Tomcat Web Site, linked in the footnote[3]. The archive `apache-tomcat-8.5.35.tar.gz` is downloaded in the `/opt/tomcat8/apache-tomcat-8.5.35` folder, and the `$TOMCAT_HOME` environment variable is created for pointing here. At this point, the following commands can be executed in the Linux shell for unzipping the downloaded file:

```
$ cd $TOMCAT_HOME
$ tar xzvf apache-tomcat-8.5.35.tar.gz
```

The command for executing the Tomcat Web Container is:

```
$ $TOMCAT_HOME/bin/catalina.sh run
```

For checking if Tomcat is correctly present in the system at the desired version, the user can type the following commands:

```
$ cd $TOMCAT_HOME/lib/
$ java -cp lib/catalina.jar org.apache.catalina.util.ServerInfo
```

The Tomcat server thus becomes reachable at the default address `http://localhost:8080`, not accessible from outside the computer where it is executed. This operation must be repeated every time the user wants to start the Web execution environment.

MySQL as a database system. In the thesis' activity, MySQL is the system chosen as relational database management, and the HeID applications were developed and tested using the `5.7.27` version of the RDBMS container. But consider that the HeID applications were developed to be flexible and configurable, so the user can install the database according to

[2]Download Apache Maven 3.6.0 - https://archive.apache.org/dist/maven/maven-3/3.6.0/binaries/

[3]Download Apache Tomcat 8.5.35 - https://archive.apache.org/dist/tomcat/tomcat-8/v8.5.35/bin/

with his preferences with the possibility to use different databases splitting the data retention; it is useful, for instance, if the user wants to store some data in an RDBMS database and others data in an In-Memory database. MySQL `5.7.27` can be retrieved from the MySQL Community website, choosing the operating system, the OS version and the desired MySQL version[4], or it can also be installed with the following commands:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository -y ppa:ondrej/mysql-5.7
$ sudo apt-get update
$ sudo apt-get install mysql-server-5.7
```

During the installation, the process will ask the user for a password that will be used for logging into the MySQL Server. After the installation, run the `mysql_secure_installation` command in order to configure some security aspects. Type on the command line

```
$ mysql_secure_installation
```

and follow the onscreen instructions:

```
Securing the MySQL server deployment.

Enter password for user root:

VALIDATE PASSWORD PLUGIN can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD plugin?

Press y|Y for Yes, any other key for No:
Using existing password for root.
Change the password for root ? ((Press y|Y for Yes, any other key for
    No) :

 ... skipping.
By default, a MySQL installation has an anonymous user,
allowing anyone to log into MySQL without having to have
a user account created for them. This is intended only for
testing, and to make the installation go a bit smoother.
You should remove them before moving into a production
environment.

Remove anonymous users? (Press y|Y for Yes, any other key for No) : y
Success.


Normally, root should only be allowed to connect from
'localhost'. This ensures that someone cannot guess at
the root password from the network.

Disallow root login remotely? (Press y|Y for Yes, any other key for No)
    : y
Success.

By default, MySQL comes with a database named 'test' that
anyone can access. This is also intended only for testing,
and should be removed before moving into a production
```

[4]Download MySQL 5.7.27 - https://dev.mysql.com/downloads/mysql/

```
environment.


Remove test database and access to it? (Press y|Y for Yes, any other key
    for No) : y
 - Dropping test database...
Success.

 - Removing privileges on test database...
Success.

Reloading the privilege tables will ensure that all changes
made so far will take effect immediately.

Reload privilege tables now? (Press y|Y for Yes, any other key for No) :
    y
Success.

All done!
```

Finally, for working with MySQL, it is necessary to login into the MySQL Server:

```
$ mysql -u root -p
```

The system will ask the user for a password, set during the installation process. If the password is correctly inserted, the user is logged inside the MySQL service, and the following screen will appear in the command prompt:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.27-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights
    reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
    statement.

mysql>
```

Create the necessary database, required by the HeID applications:

```
mysql> CREATE DATABASE ehealth_healtheid;
```

Note that some problems can occur if the *global timezone* is not set in the MySQL server, therefore, the following MySQL command must be executed, according with the user timezone:

```
mysql> SET GLOBAL time_zone = '+2:00';
mysql> quit
```

At this point, the MySQL Server is correctly configured in the system, and it is ready to be used with the HeID applications.

A.1.2 HeID Connector - Workflow Manager

As already said, this component is the junction point between eIDAS and OpenNCP, which elaborates the patient and HP actions. To configure the local environment, the user must do the following steps:

1. Check if Tomcat and the database system are correctly installed in the local machine,
2. Create the database "`ehealth_healtheid`" in MySQL or in the preferred database system,
3. Open the `$tomcat_folder/conf/server.xml` and declare the Global JNDI Resource inside the `<GlobalNamingResources>` element to enable the MySQL external configuration:

```
<Resource name="jdbc/ConfHeiD" auth="Container"
    type="javax.sql.DataSource"
    factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
    maxTotal="20" maxIdle="10"
    driverClassName="com.mysql.cj.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/ehealth_healtheid?useTimezone=true"
username="username"
password="password"
/>
```

   where `url` ,`username` and `password` values depend on the MySQL configuration.

4. Open the `$tomcat_folder/conf/context.xml` and define the Global JNDI resource link inside the Context element:

```
<ResourceLink global="jdbc/ConfHeiD" name="jdbc/ConfHeiD"
    type="javax.sql.DataSource"/>
```

The *healtheid-connector* component contains a configuration file named `default.properties` and located in $PATH/healtheid-connector/src/main/resources/, in which the user can define some configuration properties to deploy the component according to his system and his preferences: these properties are summarised as following with their description and default value. The user can change this value or can create a new external configuration file, as explained below after the properties' description.

- **server.port** is the Listening Port. This is the TCP port contained in the URL sent to the patient. It is also the port used when the component is run as a standalone JAR file (default value: `443`).
- **server.url** defines the URL basename. This is the hostname contained in the URL sent to the patient. It is also the hostname used when the component is run as a standalone JAR file (default value: `localhost`).
- **server.protocol** defines as HTTPS both the URL received by the patient as well as the endpoints exposed when the component is run as a standalone JAR file. It must be "https" (default value: `https`).
- **server.ssl.enabled** is useful to enable/disable SSL connections. If used, it should be "true". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `true`).
- **security.require-ssl** is useful to requires SSL. If used, it should be "true". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `true`).

- **server.ssl.key-store-type** defines the format used for the keystore. It could be set to JKS in case it is a JKS file. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `JKS`).
- **server.ssl.key-store** defines the path to keystone containing the certificate. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `$PATH/health-eid/ healtheid-connector/keystore/ keystore.jks`).
- **server.ssl.key-store-password** defines the password used to generate the certificate. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `password`).
- **server.ssl.key-alias** defines the alias mapped to the certificate. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `healtheid-connector`).
- **server.tomcat.remote-ip-header** defines the name of HTTP header from which the remote IP is extracted. If used, it must be "x-forwarded- for". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `x-forwarded-for`).
- **server.tomcat.protocol-header** is useful to enable setting the header of the incoming protocol. If used, it must be "x- forwarded-proto". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `x-forwarded-proto`).
- **server.tomcat.redirect-context-root** defines whether requests to the context root should be redirected by appending a / to the path. If used, it must be "true". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `true`).
- **jwt.secret** defines the shared secret (the same one that is used in eIDAS HProxy configurations). Should be redefined by the MS and configured accordingly in the other components using it (default value: `shared-secret-to-sign-and- verify-JWT-token-change-it`).
- **jwt.expiration** indicates the expiration time of the JWT token (in seconds) [600 seconds= 10 minutes] (default value: `600`).
- **redirect.to.eidasHProxy** indicates the eIDAS HProxy endpoint (default value: `http://<hostname>:<port>/eidas-hproxy/authentication`).
- **ncphproxy.url** configures the NCP HProxy endpoint (default value: `http://<hostname>:<port>/healtheid-ncp-hproxy/ncphproxy`).
- **spring.jpa.hibernate.ddl-auto** is useful to enables Hibernate to create the "ehealth_healtheid" schema tables upon deployment of the artefact (default value: `update`).
- **spring.datasource.jndi-name** externalizes JDNI configuration declared in Tomcat (default value: `java:comp/env/jdbc/ ConfHeiD`).
- **acknowledge.path** represents the classpath for the acknowledge HMTL page (default value: `classpath:acknowledge/default.html`).
- **pin.reference.version** defines the acknowledged PIN-B version. It must be redefined by the MS, according to the version of the PIN-B they are using (default value: `1.0.0.RC1`).

The following instructions allow customisation of the previous configurations using a new configuration file that will overwrite the `default.properties`. Before creating the file, the uses must add the following configuration in the Tomcat's context.xml, located in `$tomcat_folder/conf/context.xml`:

```
<Parameter name="healtheid-connector.properties"
   value="/path/to/tomcat/properties/ healtheid-connector.properties"/>
```

where:

- name: it must be `healtheid-connector.properties`
- value: absolute path to a custom properties file, e.g., can be within a newly created properties file inside Tomcat (but this is not mandatory, it can be anywhere in the filesystem, as long as the user running the HeID Connector has sufficient permissions to read it).

The configurations contained in healtheid-connector.properties file will overwrite the default configurations provided by the default.properties file included in the component artefact. Note that if this file does not exist, the default ones apply.

A.1.3 Patient I/O: the Notification Adapter

The *healtheid-notification-adapter* component is packaged as a deployable JAR file and implements an email service for demonstration purposes. The application.properties file sets the mail configuration properties to realise the service with Spring Mail:

- **spring.mail.host** defines the email server host. It should be redefined by the MS according to their email infrastructure configuration (default value: `smtp.gmail.com`).
- **spring.mail.port** defines the email server port. It should be redefined by the MS according to their email infrastructure configuration (default value: `587`).
- **spring.mail.username** defines the email authentication username. It should be redefined by the MS according to their email infrastructure configuration (default value: `testconnector2019@gmail.com`).
- **spring.mail.password** defines the email authentication password. It should be redefined by the MS according to their email infrastructure configuration (default value: `TestConnector2019`).
- **spring.mail.properties.mail.smtp.starttls.enable** is used to enable StartTLS. It should be redefined by the MS according to their email infrastructure configuration (default value: `true`).
- **spring.mail.properties.mail.smtp.starttls.required** is useful to require StartTLS. It should be redefined by the MS according to their email infrastructure configuration (default value: `true`).
- **spring.mail.properties.mail.smtp.ssl.enable** is used to enables SSL. It should be redefined by the MS according to their email infrastructure configuration.(default value: `false`).
- **spring.mail.properties.mail.smtp.auth** is used to enables email authentication. It should be redefined by the MS according to their email infrastructure configuration (default value: `true`).
- **spring.mail.properties.mail.smtp.connectiontimeout** defines the connection timeout (in ms). (default value: `5000`).
- **spring.mail.properties.mail.smtp.timeout** defines the timeout (in ms) (default value: `5000`).

- **spring.mail.properties.mail.smtp.writetimeout** used to write timeout (in ms) (default value: `5000`).
- **email.from** defines the email sender (default value: `no-reply@company.com`).

The properties contained in the Notification Adapter's application.properties file are not set (i.e., they are commented) since they are in fact globally set by the HeID-Connector component (in its default.properties file), which includes this one. However, they can be set, should the user wants to run the component locally, in an isolated way. To overwrite the Notification Adapter default configurations provided by the HeID-Connector default.properties file included within the latter, the user must provide the custom values in the `healtheid-connector.properties` file deployed within the Tomcat.

A.1.4 eIDAS HProxy

The *eidas-hproxy* component, developed as a WAR file, interacts with the *healtheid-connector* and the eIDAS Node Connector (it may require a National Adapter in a real scenario, depending in National Infrastructure). This component relies on the CEF eIDAS Sources, that can be obtained as follow:

1. Check if `maven` is correctly installed in the system,
2. Download the EIDAS-Sources 2.2.0 from CEF eIDAS Node Integration Package [73],
3. unzip the package and go in the $EIDAS-Sources/EIDAS-Parent folder,
4. run the command "`mvn clean install -DskipTests`" in the terminal,

Next step regard to configure the *eidas-hproxy* component according with the system environments and security certificates. In `$Path/eidas-hproxy/main/src/resources` there is the `default.properties` file with the following fields to configure:

- **server.port**: it defines the TCP listening port.
- **jwt.secret**: it defines the shared secret (the same one that is used in the *healtheid-connector* configuration). Should be redefined by the MS and configured accordingly in the other components using it (default value: `shared-secret-to-sign-and-verify-JWT-token-change-it`).
- **url.heidConnector.acceptPatientAuth**: it defines the url for interacting with *healtheid-connector* and communicating the result of eIDAS authentication (default value: `http://<hostname>:<port>/heidconnector/acceptPatientAuthN`).
- **eidas.path**: the file system path pointing to eIDAS configuration folder, containing Signing and Encryption settings and keystore (see below).
- **sp.metadata.url** = `http://<hostname>:<port>/eidas-hproxy/metadata`; this is the endpoint in which the component exposes his SAML metadata.
- **sp.country** = `IT`; the code of the country or organisation that provides the exit-point toward eIDAS.
- **country.metadata.url** = `http://connector-test-healtheid.polito.it/EidasNode/ServiceProvider`; the url of the eIDAS exit-point which must receive the eIDAS authentication request.
- **sp.return** = `http://<hostname>:<port>/eidas-hproxy/AuthResponse`; this is the endpoint in which the component wants to receive the SAML response from eIDAS.
- **sp.metadata.retention** = `86400`; the eIDAS connector should re-load the SP metadata page if the following time (in seconds) has elapsed.

- **encryption.algorithm.whitelist**: it contains the encryption algorithms allowed in the responses received. Each algorithm must be separated by "`;`" and no spaces are allowed in the listing.
- **signature.algorithm.whitelist**: it contains the signature algorithms allowed in the responses received. Each algorithm must be separated by "`;`" and no spaces are allowed in the listing.
- **eidas.protocol.version** = `1.1`; this is the value of eIDAS protocol version followed by the SP. When not empty, the value will be published in the SP's metadata URL.
- **eidas.application.identifier** = `CEF:eIDAS-ref:2.2.0`; This value of eIDAS protocol application identifier relative to the IdP code and version number. When not empty, the value will be published in the SP's metadata URL.
- **Metadata additional information**: SP and contact information to be displayed on the metadata page that can be defined by the `provider.name`, `sp.type`, `contact.support.email`, `contact.support.company`, `contact.support.givenname`, `contact.support.surname`, `contact.support.phone`, `contact.technical.email`, `contact.technical.company`, `contact.technical.givenname`, `contact.technical.surname` and `contact.technical.phone properties`

For configuring the communication between the *eidas-hproxy* component and the National eIDAS Connector (or National Adapter, depending on the scenario):

1. generate self-signed certificates, one for signature and one for the encryption of SAML messages
2. ask our National eIDAS Connector (or National adapter) to trust of the former certificates
3. insert the certificates in a Java Keystore at this path: `$PATH/eidas-hproxy/sp-config.local/keystore/`
4. go to `$PATH/eidas-hproxy/sp-config.local/server/sp/` and configure the `EncryptModule_SP.xml` in according with the characteristics of the certificate generated for encryption:

```
<!-- Key store configuration -->
  <entry key="keyStorePath"> ../../keystore/heid-eidas-hproxy.jks
      </entry>
  <entry key="keyStorePassword"> Password-of-the-keystore </entry>
  <entry key="keyPassword">
      Password-used-to-generate-the-encr.certificate </entry>
  <entry key="keyStoreType">JKS</entry>
```

5. go to `$PATH/eidas-hproxy/sp-config.localserver/sp/` and configure the `SignModule_SP.xml` in according with the characteristics of the certificate generated for signature:

```
<!-- signing response assertion true/false -->
        <entry key="response.sign.assertions">true</entry>
        <entry key="keyStorePath"> ../../keystore/heid-eidas-hproxy.jks
            </entry>
        <entry key="keyStorePassword"> Password-of-the-keystore </entry>
        <entry key="keyPassword">
            Password-used-to-generate-the-sign.certificate </entry>
        <entry key="issuer"> certificate-issuer </entry>
        <entry key="serialNumber"> certificate-serial-number </entry>
        <entry key="keyStoreType"> JKS </entry>
```

```
<!-- Metadata signature configuration -->
       <entry key="metadata.keyStorePath">
           ../../keystore/heid-eidas-hproxy.jks </entry>
       <entry key="metadata.keyStorePassword"> Password-of-the-keystore
           </entry>
       <entry key="metadata.keyPassword">
           Password-used-to-generate-the-sign.certificate </entry>
       <entry key="metadata.issuer"> certificate-issuer </entry>
       <entry key="metadata.serialNumber"> certificate-serial-number
           </entry>
       <entry key="metadata.keyStoreType"> JKS </entry>
```

6. set the environment variable `$eidas_config= $PATH/eidas-hproxy/sp-config.localserver/sp/`, where `$PATH` is the path in which the location of the eidas-hproxy folder,
7. go to `$PATH/eidas-hproxy/` and run “`mvn clean install`,

The generated WAR file will be located in the `$PATH/eidas-hproxy/target` directory and can be deployed on Tomcat.

Verification

Load `https://<hostname/ip>:<port>/eidas-hproxy/metadata` from a web browser and examine the resulting XML metadata page. In addition, it is possible to verify the following information:

- entityID should match the `sp.metadata.url` value, e.g.:

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
    entityID="https://heid-connector.test/eidas-hproxy/metadata"
    validUntil="2019-09-25T19:30:09.882Z">
```

- the metadata should be signed, then the `<ds:SignatureValue>` element must be present in the metadata page, e.g.:

```
<ds:SignatureValue>...</ds:SignatureValue>
```

- the public X509 Certificates appears in the XML metadata page
- the return page is set correctly if present in the `<md:AssertionConsumerService>` element, e.g.:

```
<md:AssertionConsumerService
    Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
    Location="http://heid-connector.test/eidas-hproxy/AuthResponse"
    index="0" isDefault="true"/>
<md:AssertionConsumerService
    Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
    Location="http://heid-connector.test/eidas-hproxy/AuthResponse"
    index="1"/>
```

A.1.5 NCP HProxy

The NCP HProxy component is packaged as a deployable WAR file (healtheid-ncp- hproxy). For testing purposes, it can be launched as a standalone Spring Boot JAR file containing an embedded Tomcat. This component’s behavior is managed by a set of properties within its self-contained default.properties file. The following instructions allow customisation of such configurations.

```
<ResourceLink global="jdbc/ConfHeiD" name="jdbc/ConfHeiD"
    type="javax.sql.DataSource"/>
```

Where:

- name: it must be `healtheid-connector.properties`
- value: absolute path to a custom properties file, e.g., can be within a newly created properties file inside Tomcat (but this is not mandatory, it can be anywhere in the filesystem, as long as the user running the HeID Connector has sufficient permissions to read it).

The configurations contained in ncp-hproxy.properties file will overwrite the default configurations provided by the default.properties file included in the component artefact. If this file does not exist, the default ones apply. The properties to set for install the *ncp-hproxy* component are:

- **jwt.secret** defines the shared secret (the same one that is used in the *healtheid-controller* configurations). Should be redefined by the MS and configured accordingly in the other components using it (default value: `shared-secret-to-sign-and-verify-JWT-token-change-it`).
- **jwt.expiration** indicates the JWT Expiration time (in seconds), currently not being validated. (default value: `7200`).
- **server.ssl.enabled** is useful to enable/disable SSL connections. If used, it should be "true". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `true`).
- **security.require-ssl** is useful to requires SSL. If used, it should be "true". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `true`).
- **server.ssl.key-store-type** defines the format used for the keystore. It could be set to JKS in case it is a JKS file. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `JKS`).
- **server.ssl.key-store** defines the path to keystone containing the certificate. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `$PATH/health-eid/healtheid-connector/keystore/keystore.jks`).
- **server.ssl.key-store-password** defines the password used to generate the certificate. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `password`).
- **server.ssl.key-alias** defines the alias mapped to the certificate. Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `healtheid-hproxy`).
- **server.tomcat.remote-ip-header** defines the name of HTTP header from which the remote IP is extracted. If used, it must be "x-forwarded- for". Only used in the case the component is run as a standalone JAR file. Otherwise it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `x-forwarded-for`).

- **server.tomcat.protocol-header** is useful to enable setting the header of the incoming protocol. If used, it must be "x- forwarded-proto". Only used in the case the component is run as a standalone JAR file. Otherwise, it can be ignored, given that such configuration is provided by the application server where it is deployed (default value: `x-forwarded-proto`).

Even though the NCP HProxy is configured to use Spring Security to increase security, at the moment, its only endpoint is freely exposed (i.e., no JWT or any other kind of token validation is performed, as described previously in the jwt.* properties). However, the component is prepared to be configured accordingly.

NCP HProxy default implementation

With the default implementation provided by HEALTHeID, the NCP HProxy must be deployed in the NCP infrastructure, since the default National Adapter depends on the EPSOS_PROPS_PATH environment variable (deeply tied to the OpenNCP reference implementation [74]). This variable must be available (e.g., via the Tomcat's `$TOMCAT_PATH/bin/setenv.sh` file). Additionally, the JNDI resource `jdbc/ConfMgr`, demanded by the OpenNCP components, must be configured in the Tomcat where the NCP HProxy is deployed (in `$TOMCAT_PATH/conf/context.xml` and `$TOMCAT_PATH/conf/server.xml`).

context.xml

```
<ResourceLink global="jdbc/ConfMgr" name="jdbc/ConfMgr"
    type="javax.sql.DataSource"/>
```

sever.xml

```
<GlobalNamingResources>

<Resource name="UserDatabase" auth="Container"
    type="org.apache.catalina.UserDatabase" description="User database that
    can be uptated and saved"
    factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
    pathname="conf/tomcat-users.xml"
/>

<Resource name="jdbc/ConfMgr" auth="Container" type="javax.sql.DataSource"
    factory="com.zaxxer.HikariJNDIFactory" singleton="true" minimumIdle="2"
    maximumPoolSize="5" connectionTimeout="300000"
    dataSourceClassName="com.mysql.jdbc.jdbc2.optional.MysqlDataSource.Driver"
    dataSource.serverName="server" datasource.port="3306"
    datasource.darabaseName="database" username="user" password="password"
/>

</GlobalNamingResources>
```

For the previous configuration, a JDBC Connection Pool such as HikariCP must be available. This is achieved by placing the HikariCP-2.6.3.jar, slf4j-api-1.7.25.jar and mysql-connector-java.jar (e.g., version 5.1.48) in the `$TOMCAT_PATH/lib` folder (same versions used by the OpenNCP are used here for ease of installation).

Following are the Maven artefacts declaration to help their identification in the Maven Central repository:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.48</version>
</dependency>
```

```

<dependency>
<groupId>com.zaxxer</groupId>
  <artifactId>HikariCP</artifactId>
  <version>3.1.0</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.25</version>
</dependency>
```

In case another database provider is used, a different JAR file rather than the MySQL one must be used and the JNDI resource must be configured accordingly. This default implementation of the NCP HProxy looks for the property PORTAL_CLIENT_CONNECTOR_URL of the OpenNCP properties schema (ehealth_properties), which should point to the OpenNCP Client Connector deployed at NCP-B (as in a typical OpenNCP installation). For more information about OpenNCP installation see [74].

NCP HProxy national implementation

With a national implementation, it may be deployed separately.

Appendix B

Programmimg Manual

This appendix describes the contents of the HeID Connector web application, the eidas-hproxy module, the NCP HProxy (developed by SPMS) and the Notification Adapter, designed and developed as part of the thesis project. Table B.1 describes the meaning of the constants mentioned later in the discussion. These are created to simplify the description of the path of each resource belonging to the sources of the developed software, without constraining them to a particular execution environment. All the paths are relative to the root of the directory tree that makes up the thesis work and are indicated in the Linux syntax, with the separator character /, or slash.

Variable	**Value**	**Meaning**
$SRC_DIR	sources folder	Directory containing the components of the code developed in the thesis project.
$EU_DIR	$SRC_DIR/healtheid-connector/src/main	Source directory of the HeID Connector Web Application.
$EU_JDIR	$EU_DIR/java/eu/europa/ec/healtheid	Java source directory of the HeID Connector Web Application.

Table B.1. Environment variables for understanding the programmer's manual.

B.1 HeID Connector: source code, resources and business logic

The *HeID Connector* module is is implemented as a Maven *webapp* project, identified by **groupId** equal to **eu.europa.ec.healtheid** and **artifactId** as **healtheid-parent**. The connector is distributed as a WAR archive and it is based on the JDK 1.8 version of the Java API.

B.1.1 Class: Encounter

This class represents the Encounter between the Healthcare Professional and the Patient. Table B.2 shows the attributes belonging to each instance of this class. There are a public setter and getter methods for each attribute, as well as a constructor without arguments. Note that there is a "*@Entity*" annotation, which let Spring know that *Encounter* class represents a table in the configured database with the *token* attribute as the primary key of the entity; this latter is annotated as "*@Id*".

Package: eu.europa.ec.healtheid.models

Class: Encounter

File: $EU_JDIR/models/Encounter.java

Attribute	**Data type**	**Meaning**
patientEmail	String	The email address to contact the patient for sending encounter link or some notifications.
patientMobilePhone	String	The mobile phone number to contact the patient for sending encounter link or some notifications via SMS.
patientCountryCode	String	Code which identifies the patient's country, e.g. "IT" code means that patient is Italian citizen.
token	String	JSON Web Token represented as String value in the format *aaa.bbb.ccc.* It is the the encounter ID.
consent	Boolean	Status of patient consent. If set to false, the patient did not give his consent.

Table B.2. Content of the Encounter Java object defined in the HeID Connector project.

B.1.2 Interface: EncounterRepository

The interface defines a repository to store the *Encounter* B.1.1 in the database by its encounterID value. Note that this interface is not implemented and its working code will be automatically generated from its signature by Spring Data JPA. It extends the CrudRepository (add reference).

Package: eu.europa.ec.healtheid.repository

Class: EncounterRepository

File: $EU_JDIR/models/EncounterRepository.java

```
@Transactional
public interface EncounterRepository extends CrudRepository<Encounter,
    String> {
}
```

The "*@Transactional* " annotation delegates to Spring to performs database transactions.

B.1.3 Class: PatientData

PatientData represents the collection of patient attributes recovered from eIDAS after the patient authentication and from the additional data action performed if some information are missing. The class has the "*@Entity*" annotation and the set of java attributes is shown in table B.3. Note that *encounterID* attribute has the "*@Id*" annotation and symbolises the primary key of the entity.

Package: eu.europa.ec.healtheid.models

Class: PatientData

File: $EU_JDIR/models/PatientData.java

Attribute	Data type	Meaning
encounterID	String	JSON Web Token represented as String value in the format *aaa.bbb.ccc*.
dataReady	Boolean	This value represents the status of patient data. When patient is authenticated and the system knows the patientID, this boolean value is true .
demographics	List<Demographics>	Code which identifies the patient's country, e.g. "IT" code means that patient is Italian citizen.

Table B.3. Content of the PatientData Java object defined in the HeID Connector project.

B.1.4 Interface: PatientDataRepository

This interface defines a repository to store *PatientData* B.1.3 in the database by its encounterID. Note that this interface is not implemented and its working code will be automatically generated from its signature by Spring Data JPA. It extends the CrudRepository (add reference).

Package: eu.europa.ec.healtheid.repository

Class: PatientDataRepository

File: $EU_JDIR/models/PatientDataRepository.java

```
@Transactional
public interface PatientDataRepository extends CrudRepository<PatientData,
    String> {}
```

The "*@Transactional* " annotation delegates to Spring to performs database transactions.

B.1.5 Class: NotificationData

This class represents the NotificationData that will be sent by HeID Client to inform the HeID Connector about what is happening in the OpenNCP World (XCPD, EP, PS,). The set of java attributes is shown in table B.4.

Package: eu.europa.ec.healtheid.models

Class: NotificationData

File: $EU_JDIR/models/NotificationData.java

Attribute	Data type	Meaning
notificationType	NotificationType	Enumerator which defines the possibles notification types.
encounterID	String	JSON Web Token represented as String value in the format *aaa.bbb.ccc*.

Table B.4. Content of the NotificationData Java object defined in the HeID Connector project.

B.1.6 Class: NotificationType

Class belonging to package *eu.europa.ec.healtheid.models.enumerator* which defines some static values for the kind of notification that the HeID Connector can send to the patient. The enum values are defined as follow:

```
public enum NotificationType {
  ED,
  EP,
  PS,
  XCPD,
  CONSENT
}
```

The *ED* value is uses for representing a eDispensation notifications, *EP* for ePrescriptions, *PS* for Patient Summaries, *XCPD* for notifying the end of OpenNCP Patient Discovery and finally *CONSENT* for a notification about the consent mechanism.

B.1.7 Class: ErrorResponse

The *ErrorResponse* class provides some attributes (see Table B.5) in which store and then notify a potential error that can occurs at the application level.

Package: eu.europa.ec.healtheid.models

Class: ErrorResponse

File: $EU_JDIR/models/ErrorResponse.java

Attribute	**Data type**	**Meaning**
status	HttpStatus	Enumeration of HTTP status code.
error_code	String	Error code of a potential issue that can occur in the HeID Connector Web Application
description	String	Short description of the error.

Table B.5. Content of the ErrorResponse Java object defined in the HeID Connector project.

B.1.8 Class: PatientAcknowledge

This class represents the PatientAcknowledge that will be stored in a database. The class is marked with the "*@Entity*" annotation to inform the Spring framework that it represents a table inside the HeID Database. The JWTtoken, i.e. the encounterID, is used as the primary key as usual; the other fields are depicted in table B.6.

Package: eu.europa.ec.healtheid.models

Class: PatientAcknowledge

File: $EU_JDIR/models/PatientAcknowledge.java

B.1.9 Interface: AcknowledgeRepository

This interface defines a repository to store *PatientAcknowledge* B.1.8 in the database by its encounterID. Note that this interface is not implemented and its working code will be automatically generated from its signature by Spring Data JPA.It extend the CrudRepository (add reference).

Attribute	Data type	Meaning
JWTtoken	String	JSON Web Token represented as String value in the format *aaa.bbb.ccc*.
consent	Boolean	Status of patient acknowledgment. If set to false, the patient did not give his acknowledgment.
timestamp	java.util.Date	The moment in which the patient performs the acknowledgment action.
PINreference	String	The reference to the Patient Information Notice, which was shown at the patient.
PatientID	String	The Patient Identifier.

Table B.6. Content of the PatientAcknowledge Java object defined in the HeID Connector project.

Package: eu.europa.ec.healtheid.repository

Class: AcknowledgeRepository

File: $EU_JDIR/models/AcknowledgeRepository.java

```
@Transactional
public interface AcknowledgeRepository extends
    CrudRepository<PatientAcknowledge, String> {}
```

The "*@Transactional* " annotation delegates to Spring to performs database transactions.

B.1.10 Interface: NcpHProxyService

Interface for connecting the HeID Connector to the NCP HProxy component in order to get the OpenNCP Configuration of patient's country. The class is annotated as Spring "*@Component*" which marks the class as a bean so that Spring can detect it and pull it into the ApplicationContext. In the following is described the unique method of this class.

Method: getCountryConfiguration

Package: eu.europa.ec.healtheid.services

Class: NcpHProxyService

File: $EU_JDIR/services/NcpHProxyService.java

```
public HealtheidCountryConfiguration getCountryConfiguration(String country)
    throws HealtheidGetCountryConfigurationException;
```

This method receives the patient country code and returns the HealtheidCountryConfiguration object which represents the configuration of the remote country for retrieving patient data. The implementation of the method can be found in the *NcpHProxyServiceImpl.java* class, located in the same package. It implements a REST call toward the NCP HProxy component on the endpoint

https//:<hostname:port >/healtheid-ncp-hproxy/ncphproxy/`{country}`

where the `{country}` parameter is the String value received by the method as input. Instead the $< \text{hostname} : \text{port} >$ depends on the NCP HProxy deployment; note that the endpoint value is configurable in the default.properties [add reference] file and injected inside the *NcpHProxyServiceImpl.java* instance by Spring in the String *ncphproxyUrl* variable which has the " *@Value*" annotation. A *HealtheidGetCountryConfigurationException* is raised if something wrong happens, and the health country configuration can not be retrieved.

B.1.11 Interface: PatientConnector

Interface for connecting the HeID Connector to the patient toward the Notification Adapter component, which implements the SMS/email communication to send the notifications. The class is annotated as Spring "*@Component*" which marks the class as a bean so that Spring can detect it and pull it into the Application-Context. In the following the two methods of this class:

Method: sendLinkToPatient

Package: eu.europa.ec.healtheid.services

Class: PatientConnector

File: $EU_JDIR/services/PatientConnector.java

```
public boolean sendLinkToPatient(String link, String phoneOrEmail);
```

This method contacts the Notification Adapter, i.e. `HealtheidNotifier` component, and receives as input parameters the link, generated by the Workflow Manager, and the email or the mobile phone of the patient. It returns a boolean value: `true` if the link is correctly sent to the patient by the Notification component, `false` otherwise. The implementation of the method can be found in the *PatientConnectorImpl.java* class, located in the same package.

Method: sendNotificationToPatient

Package: eu.europa.ec.healtheid.services

Class: PatientConnector

File: $EU_JDIR/services/PatientConnector.java

```
public boolean sendNotificationToPatient(NotificationType type, String
    phoneOrEmail);
```

This method contacts the Notification Adapter, i.e. *HealtheidNotifier* component, receiving as input parameters the NotificationType B.1.6 object, which contains the kind of advice to send, and the email or the mobile phone of patient. It returns a boolean value: `true` if the notification is correctly sent to patient by the *HealtheidNotifier* component, `false` otherwise. The implementation of the method can be found in the *PatientConnectorImpl.java* class, located in the same package.

B.1.12 Class: HeiDClientController

`HeiDClientController` class performs the role of an entry/exit point for the communication with the `HeID Client` component, located in the OpenNCP portal. This class handles all the incoming requests and contacts the Workflow Manager to elaborate on the corresponding responses. It is characterised by the Spring *@RestController* annotation and provides three REST endpoints, reported in the table B.7, that the `HeID Client` can call for creating the encounter, polling the availability of patient data and sends relevant updates to forward toward the patient informing him of what is happening in the system. These endpoints are manage by the following methods:

Method: createEncounter

Package: eu.europa.ec.healtheid.controllers

Class: HeIDClientController

File: $EU_JDIR/controllers/HeIDClientController.java:58

```
public ResponseEntity<Encounter> createEncounter(@RequestBody Encounter
    encounter){}throws Exception;
```

Endpoint	**Request Parameters**	**Response**
POST /encounter/createEncounter	*Encounter* (B.1.1) object created by the HeID Client, which contains the patient information about country of origin and email/SMS. It is requested in the HTTP Body.	200 OK - the connector responds with the same Encounter object, with the field *encounterID* filled with the JWT token.
POST /encounter/requestPatient-Data	*encounterID* value in the HTTP Authorization Header. It is used as authorisation token.	200 OK - If the patient data are ready 204 No Content - If patient has not yet completed the authentication
POST /encounter/receiveNotice	*encounterID* value in the HTTP Authorisation Header. *NotificationData* (B.1.5) object which contains the kind of notice to send toward patient is required in the HTTP Body	200 OK - The notification is correctly sent to patient

Table B.7. HeID connector endpoints to handle HeID Client requests.

Response example:

`HTTP Status code: 200 OK`

`Content-Type: application/json`

```
{
        "patientEmail": "italian.patient@email.it "

        "patientMobilePhone" : "323 33 33 333"

        "patientCountryCode" : "IT"

        "token" :
            "eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIxYzE0YTE5MC0wZmFjLTRlMTktYjk0Yi0yZ
   GViNzg3ODYwZjkiLCJleHAiOjE1NjMyMzYzMjEsInN1YiI6IklUIn0.foIhtBB2qH-
   vyKGOFcivYc54VeAVhg6GtPgaoEi68I"

        "consent" : "false"
}
```

Method responsible for receiving the HTTP encounter request from HeID Client. It receives the `Encounter` object as a parameter, which contains the necessary patient's data to start the encounter process. The method calls the `WorkflowManager` which fulfil the data object with the `encounterID` and sets the `consent` value to `false`. The "*@RequestBody*" annotation indicates that the input parameter should be bound into the body of the web request.

Method: requestPatientAuthNData

Package: eu.europa.ec.healtheid.controllers

Class: HeIDClientController

File: $EU_JDIR/controllers/HeIDClientController.java:94

```
public ResponseEntity<PatientData>
    requestPatientAuthNData(@RequestHeader("Authorization") String
    encounterID) throws PatientDataNotReadyException,
    PatientNotFoundException,Exception;
```

Response example:

`HTTP Status code: 200 OK`

`Content-Type: application/json`

```
{
        "encounterID" :
            "eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIxYzE0YTE5MC0wZmFjLTR1MTktYjk0YiOyZ
   GViNzg3ODYwZjkiLCJleHAiOjE1NjMyMzYzMjEsInN1YiI6IklUIn0.foIhtBB2qH-
   vyKGOFcivYc54VeAVhg6GtPgaoEi68I"

        "dataReady" : "true"

        "demographics" : {
                "Demographics" : [
                {
                        "friendlyName" : "Surname"
                        "userValue" : "Rossi"
                        ...
                        ...
                },
                {
                        "friendlyName" : "Name"
                        "userValue" : "Giovanni"
                        ...
                        ...
                },
                {
                        "friendlyName" : "patientID"
                        "userValue" : "RSSGVNN12G33B345A"
                        ...
                        ...
                },
                {
                        other attribute element
                },
                etc...
                ]
        }

}
```

This method handles the request about patient data availability. The data are available when the patient completed his authentication and the `WorkflowManager` (WM) knows the PatientID. So this method asks to WM for patient data and exploits the `200 OK` and the `204 No Content` HTTP status code to inform the client. The "*@RequestHeader*" annotation indicates that a method parameter should be bound to a web request header, i.e. the encounterID, which is used both to identify the encounter and as authorisation token to legitimate the request.

Method: receiveNotice

Package: eu.europa.ec.healtheid.controllers

Class: HeIDClientController

File: $EU_JDIR/controllers/HeIDClientController.java:119

```
public ResponseEntity<Object> receiveNotice(@RequestHeader("Authorization")
    String encounterID, @RequestBody NotificationData notice, ModelMap model)
    throws Exception;
```

Response example:

```
HTTP Status code: 200 OK
Content-Type: No Content
```

This method receives the kind of notification that the WM forwards to the Patient for informing him about the various steps of OpenNCP process or his consent status. The "*@RequestHeader*" annotation indicates that a method parameter should be bound to a web request header, i.e. the encounterID, which is used both to identify the encounter and as authorisation token to legitimate the request.

B.1.13 Class: PatientController

PatientController class defines the method that handles the Patient interactions and drives his action for performing the authentication and storing the consent decision. The "*@Controller*" indicates to Spring that this class is a web controller, which provides different endpoints, reported in the table B.8. The methods of this class are described as follow:

Endpoint	**Request Parameters**	**Response**
GET /patientEncounter/acceptEncounter/`{token}`	the encounterID, extracted frome the link	Redirect the patient toward the eIDAS HProxy.
GET /patientEncounter/additionalPatientData	`String token`: the encounterID	Redirect to the AdditionalData Form if the eIDAS attributes are not enough (e.g. PatientID is missing)
POST /patientEncounter/additionalPatientData	`String token`: the encounterID `SearchFieldsForm searchFieldsForm` : the set of attributes added by hand because not provided by the eIDAS attribute set.	Redirect to Acknowledgement Screen or again in the AdditionalData Screen if something is missing.
POST /patientEncounter/patientAcknowledge	`String token`: the encounterID `Encounter encounter`: the encounter data structure.	Shows the Patient Information Notice.
POST /patientEncounter/acknowledge	`Encounter encounter`: the encounter data structure	Shows the result of the Acknowledgment process.

Table B.8. HeID connector endpoints to drive patient actions.

Method: acceptEncounter

Package: eu.europa.ec.healtheid.controllers

Class: PatientController

File: $EU_JDIR/controllers/PatientController.java:71

```
public ModelAndView acceptEncounter(ModelMap model, @PathVariable(value =
    "token") String token);
```

This method is responsible for accepting the Encounter. The endpoint receives the encounterID (JWT Token) that is accepted as "*@PathVariable*". When the patient clicks on the link received by email, he will be redirected to this endpoint. Here the `WorkflowManager` is called to validate the encounterID.

Method: additionalPatientData

Package: eu.europa.ec.healtheid.controllers

Class: PatientController

File: $EU_JDIR/controllers/PatientController.java:91

```
public ModelAndView additionalPatientData( ModelMap model,
    @ModelAttribute(value="token") String token) throws
    HealtheidGetCountryConfigurationException;
```

This method responsible for showing the AdditionalData Screen after the eIDAS authentication, if needed. It contacts the `NCP HProxy` to check if the eID Schema provided by the Patient Country contains all the necessary attributes; if not, this method shows to patient the AdditionalData Screen, otherwise it shows the Acknowledgment Screen. The endpoint receives the encounterID (token) as "*@ModelAttribute*". The HealtheidGetCountryConfigurationException is raised if some problems occur during the NCP HProxy interaction.

Method: postAdditionalPatientData

Package: eu.europa.ec.healtheid.controllers

Class: PatientController

File: $EU_JDIR/controllers/PatientController.java:120

```
public ModelAndView postAdditionalPatientData(@ModelAttribute(value =
    "token") String token, @ModelAttribute(value = "searchFieldsForm")
    SearchFieldsForm searchFieldsForm, ModelMap model) throws
    PatientNotFoundException, HealtheidGetCountryConfigurationException;
```

This method accepts the inputs of the AdditionalData Screen, that are the attributes missing from eIDAS authentication and inserted by hand; this set of attributes is contained in the `SearchFieldsForm searchFieldsForm` object.

Method: patientAcknowledge

Package: eu.europa.ec.healtheid.controllers

Class: PatientController

File: $EU_JDIR/controllers/PatientController.java:146

```
public String patientAcknowledge(ModelMap model,
    @ModelAttribute(value="token") String token, @ModelAttribute(value =
    "encounter") Encounter encounter) throws IOException;
```

This method redirect the Patient to acknowledge Screen, showing to him the Patient Information Notice and redirect the user to `/patientEncounter/acknowledge` to store the result of the acknowledge in the database.

Method: acknowledge

Package: eu.europa.ec.healtheid.controllers

Class: PatientController

File: $EU_JDIR/controllers/PatientController.java:170

```
public String acknowledge(@ModelAttribute(value = "encounter") Encounter
    encounter, SessionStatus status) throws Exception;
```

B.1.14 Class: EidasHProxyController

This class is responsible for handling the responses coming from the `eIDAS HProxy` component. This messages contains the attributes obtained from the eIDAS authentication performed by the patient. A unique endpoint is defined to receive this information:

/heidconnector/acceptPatientAuthN

that accepts the eIDAS attributes as a JSON object that is validated against the `Map<String,String>` object. The map `key` is the friendly name of the attribute while the map `value` correspond to the attribute value.

Method: acceptPatientAuthNData

Package: eu.europa.ec.healtheid.controllers

Class: PatientController

File: $EU_JDIR/controllers/PatientController.java:146

```
public String acceptPatientAuthNData(HttpServletRequest request,
    @RequestBody Map<String,String> eidasAttributes, RedirectAttributes
    redirectAttributes, ModelMap model) throws
    HealtheidGetCountryConfigurationException;
```

This method accepts the eIDAS attributes coming from the eIDAS HProxy and passes these attributes to the Workflow Manager.

B.1.15 Class: Workflow Manager

This class is the core of the HeID Connector: it drives the Health-eID workflow, validating requests, taking decisions, conducting the invocation of the other components to precess requests and storing useful data in the database. It isolates the business logic from input and knows how to give specific tasks to other components. It is responsible to create and validate the encounter, to redirect the patient toward eIDAS, to map the eIDAS attributes in OpenNCP attributes, to ask the acknowledge and store the result. The following methods implements all these functionalities relying on other classes defined in the *package* `eu.europa.ec.healtheid.utils`.

Method: createEncounter

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:82

```
public Encounter createEncounter(Encounter encounterRequest)
```

This method is responsible to create the encounterID, as a JSON Web Token, relying on the `JwtTokenUtils` B.1.19 class, and generates the link to send to the patient. It accepts an Encounter B.1.1 object as input parameter, which contains the patient email and country, so then it fills the received data structure with the encounterID and sets the consent value to `false`. It calls the Notification Adapter to invite the patient at the encounter and returns the modified `encounterRequest`

Method: getCountryConfiguration

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:118

```
public HealtheidCountryConfiguration getCountryConfiguration(String token)
    throws HealtheidGetCountryConfigurationException;
```

Method responsible for contacting the NCP HProxy in order to obtain the specific country configuration. It receives the encounterID and, relying on the `JwtTokenUtils` B.1.19 class, it reads the interested country code.

Method: acknowledgeStore

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:131

```
public void acknowledgeStore(String jwtToken, String PINReference) throws
    Exception;
```

This method is responsible for storing the Acknowledge in DB. It receives the `jwtToken` which identifies the encounter and the `PINReference` which points to the specifc Patient Information Notice shown to patient.

Method: eIDASredirection

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:226

```
public ModelAndView eIDASredirection(ModelMap model, String encounterID);
```

This methods is responsible to forward the patient toward the eIDAS HProxy. When it is called, the method store a PatientData B.1.3 object in the database, with the encounterID passed as input parameter, and setting the `dataReady` value to `false`. This means that the encounter is accepted by the patient but he has not completed the authentication process, so his attributes are not yet available.

Method: checkIfPatientIsAuthenticated

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:161

```
public PatientData checkIfPatientIsAuthenticated(String encounterID) throws
    PatientDataNotReadyException, PatientNotFoundException;
```

This method checks if the patient has correctly completed the authentication through eIDAS. It raises the `PatientNotFoundException` if the `encounterID` does not exist, otherwise he recovers the *PatientData* from the *PatientDataRepository* and raises the `PatientDataNotReadyException` if the `dataReady` value is `false`. It returns the *PatientData* object identified by the `encounterID` if the `dataReady` value is `true`.

Method: notifyPatient

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:199

```
public boolean notifyPatient(NotificationData notice) throws Exception;
```

This method trigger the patient notification's action that will be delegated to the HealtheID Notification component. It receives the NotificationData B.1.5 which contains the reason of the notice and the encounterID. After retrieving the patient SMS/email from database, it interacts with the *PatientConnector* to inform the patient.

Method: processEncounterData

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:249

```
public String processEncounterData(String token, HttpServletRequest request,
    Map<String, String> eidasAttributes, RedirectAttributes
    redirectAttributes;
```

This method receives the `eidasAttributes` from the eIDAS HProxy. Its task is to maps this data, that follows the eIDAS SAML attribute profile [31], in the OpenNCP attributes. In addition, the method contacts the `NCP HProxy` to retrieve the *HealtheidCountryConfiguration* and checks if the `eidasAttributes` contains all the necessary data to start with the OpenNCP session. If yes, the data are stored in the *PatientDataRepository* with the `dataReady` value set to `true` and the flow goes on with the acknowledgment's require. Otherwise, they are stored with the `dataReady` set to `false` and the flow is redirected to the additionalPatientData process.

Method: putAdditionalPatientData

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:313

```
public void putAdditionalPatientData(String encounterID, SearchFieldsForm
    searchFieldsForm);
```

This method handles the additional data inserted by hand that are inside the `searchFieldsForm` variable.

Method: updateAcknlowledgement

Package: eu.europa.ec.healtheid.management

Class: Workflow Manager

File: $EU_JDIR/controllers/Workflow Manager.java:373

```
public void updateAcknlowledgement(NotificationData notice) throws
    AcknowledgmentNotFoundException, PatientNotFoundException;
```

This method is called for updating the acknowledgment with the corresponding patient decision and trigger the Notification Adapter for notifying the result to the patient.

B.1.16 Class: JwtAuthenticationEntryPoint

This class implements the Spring AuthenticationEntryPoint interface, which provided the *commence* method useful for realising an authorisation service based on HTTP headers.

Method: commence

Package: eu.europa.ec.healtheid.security

Class: JwtAuthenticationEntryPoint

File: $EU_JDIR/security/JwtAuthenticationEntryPoint.java:18

```
@Override
public void commence(HttpServletRequest request, HttpServletResponse
    response, AuthenticationException authException) throws IOException{
       response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
          "Unauthorized");
}
```

This method is responsible to commence an authentication scheme. It is called at any HTTP message and sends a *Unauthorised* response if the message does not come form an authorised entity.

B.1.17 Class: JwtAuthenticationTokenFilter

This class extends the Spring OncePerRequestFilter class which configures a granular filter that reads and checks the content of the HTTP Authorisation header.

Method: doFilterInternal

Package: eu.europa.ec.healtheid.security

Class: JwtAuthenticationTokenFilter

File: $EU_JDIR/security/JwtAuthenticationTokenFilter.java:27

```
@Override
protected void doFilterInternal(HttpServletRequest request,
    HttpServletResponse response, FilterChain filterChain) throws
    ServletException, IOException;
```

This method extract the content of Authorisation Header and validates it according to the chosen policies and decided if a message should be read by the internal logic or not.

B.1.18 Class: FileUtils

This class contains a single method used to load a file as a string inside the application.

Method: loadFileAsString

Package: eu.europa.ec.healtheid.utils

Class: FileUtils

File: $EU_JDIR/security/FileUtils.java:27

```
public String loadFileAsString(String filePath) throws IOException;
```

This method is used to load a file inside the web application. In the specific, it is called to load the PIN acknowledgment file that must be shown to the patient.

B.1.19 Class: JwtTokenUtils

This class contains the methods which handle the JSON Web Token.

Method: validateToken

Package: eu.europa.ec.healtheid.utils

Class: JwtTokenUtils

File: $EU_JDIR/security/JwtTokenUtils.java:36

```
public boolean validateToken(String token);
```

This method validates the `token` checking the `signature`, re-computing the

```
HMAC-SHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload),
    sha-256(secret))
```

and comparing it with the "*ccc*" value. If the values match, it reads the token to check the `exp` expiration value. If the token is valid, the method returns `true`; otherwise, it returns `false`.

Method: createToken

Package: eu.europa.ec.healtheid.utils

Class: JwtTokenUtils

File: $EU_JDIR/security/JwtTokenUtils.java:61

```
public String createToken(Encounter encounterRequest);
```

This method is responsible to create the JSON Web Token relying on the `io.jsonwebtoken` library. The token is created exploiting the patient country code that can be found in the `encounterRequest` and adding an expiration time, configurable in the *default.properties* configuration file.

Method: readCountry

Package: eu.europa.ec.healtheid.utils

Class: JwtTokenUtils

File: $EU_JDIR/security/JwtTokenUtils.java:94

```
public String readCountry(String token);
```

This method reads the patient country code from the JWT token, relying on the `io.jsonwebtoken`, and returns the readed value as a String.

B.1.20 Class: SearchFieldsFormUtils

This class is responsible for processing the fields which are loaded or inserted in the Views, organising them in order to be handled in a correct way (developed by SPMS).

Method: prepareForView

Package: eu.europa.ec.healtheid.utils

Class: SearchFieldsFormUtils

File: $EU_JDIR/security/SearchFieldsFormUtils.java:10

```
public static SearchFieldsForm prepareForView(SearchFieldsForm
    searchFieldsForm);
```

Method: processViewReturn

Package: eu.europa.ec.healtheid.utils

Class: SearchFieldsFormUtils

File: $EU_JDIR/security/SearchFieldsFormUtils.java:14

```
public static SearchFieldsForm processViewReturn(SearchFieldsForm
    searchFieldsForm);
```

B.1.21 Class: HealtheidConnectorConstants

This class is a collection of static variables. These static variables are used to define the components' endpoint and the friendly name of the Patient Identification Code used in the OpenNCP environment.

Package: eu.europa.ec.healtheid.config

Class: HealtheidConnectorConstants

File: $EU_JDIR/config/HealtheidConnectorConstants.java

B.1.22 Class: HealtheidExceptionHandler

This class is responsible to catch and manage all the exceptions that can occurs when the *healtheid-connector* receives some requests; it is annotated with "*@ControllerAdvice*" which is a specialisation of the "*@Controller*" annotation and it can be seen as an interceptor of exceptions thrown by methods annotated with "*@RequestMapping*".

Method: handleAllException

Package: eu.europa.ec.healtheid.config

Class: HealtheidExceptionHandler

File: $EU_JDIR/config/HealtheidExceptionHandler.java:19

```
@ExceptionHandler({Exception.class})
public ResponseEntity<Object> handleAllException(Exception ex);
```

This method catch all the exception that do not have a custom exception catcher. It returns a HTTP response with the `500 - Internal Error` status to signalling to client that the *healteid-connector* had an internal problem during the processing of the request.

Method: handlePatientDataNotReady

Package: eu.europa.ec.healtheid.config

Class: HealtheidExceptionHandler

File: $EU_JDIR/config/HealtheidExceptionHandler.java:31

```
@ExceptionHandler({PatientDataNotReadyException.class})
protected ResponseEntity<Object>
    handlePatientDataNotReady(PatientDataNotReadyException ex)
```

This method catches all the `PatientDataNotReadyException` exceptions and returns a HTTP message with `204 - No Content` status to signalling to client that the patientData are not ready at the moment, so the client must try later. Note that other custom methods can be defined to handle custom exceptions and communicate events to client,

B.1.23 Class: PersistenceJNDIConfig

This is a configuration class which contains some "*@Bean*" methods responsible for enabling the Spring container to configure Hibernate as a JPA provider with a JNDI datasource.

Method: entityManagerFactory

Package: eu.europa.ec.healtheid.config

Class: PersistenceJNDIConfig

File: $EU_JDIR/config/PersistenceJNDIConfig.java:44

```
public LocalContainerEntityManagerFactoryBean entityManagerFactory() throws
    NamingException
```

This method sets up a shared JPA EntitiyManagerFactory to lookup at the external JNDI datasource defined by the **spring.datasource.jndi-name** property in the `default.properties` configuration file.

B.1.24 Class: WebSecurityConfig

This is a configuration class used for configuring Spring Security in the web application; it extends the WebSecurityConfigurerAdapter base class. It allows customisation by overriding methods. **Method: configure**

Package: eu.europa.ec.healtheid.security.config

Class: WebSecurityConfig

File: $EU_JDIR/security/config/WebSecurityConfig.java:41

```
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception;
```

This method configures the security at HTTP level, realising a URL-based access control service and filtering the requests checking the authorisation token. Note that the /createEncounter and the patient link are exiles by the filter controls.

B.1.25 Exceptions

In the *package* `eu.europa.ec.healtheid.exception` the following class are defined, in order to trace some errors that can be occurs during the execution.

- PatientNotFoundException, which will triggered if no one `encounterID` exists in the PatientDataRepository.
- PatientDataNotReadyException, which will triggered if the `DataReady` value is set to `false` after the recovering of PatientData from the PatientDataRepository.
- HealtheidGetCountryConfigurationException, which will triggered if some issue occurs during the communication or the message processing with the NCP HProxy.
- AcknowledgmentNotFoundException, which will triggered if no one `encounterID` exists in the AcknowledgeRepository.

Note that all the exceptions are catched and managed by the HealtheidExceptionHandler B.1.22 in the *package* `eu.europa.ec.healtheid.conf`:

B.2 HeID Notification Adapter: source code, resources and business logic

This component is responsible for contacting the patient by email/SMS: the necessary network infrastructure, email configurations and the sending SMS feature must be prepared in advance by the MS (e.g., configuring email relay server; network access between the HEALTHeID-Connector infrastructure and email infrastructure). However, an exemplary demonstration of a possible implementation can be found in the sources of the project, inside the *healtheid-notification-adapter* folder. This demonstration gives a possible implementation of the sending email feature while the SMS feature is not provided.

B.2.1 Interface: HealtheidNotifier

Method: notify

$PATH_DIR: $SRC_DIR/healtheid-notification-adapter-interface/src/main

$PATH_JDIR : $PATH_DIR/java/eu/europa/ec/healtheid/

Package: eu.europa.ec.healtheid.notification.adapter.service

Class: HealtheidNotifier

File: $PATH_JDIR/notification/adapter/service/HealtheidNotifier.java

```
public HealtheidNotification notify(HealtheidNotification
    healtheidNotification);
```

B.2.2 Class: HealtheidNotification

This class includes the characteristics of the notification, such as the Object of the email, the message, the receiver, and a boolean which marks if the notice was correctly sent to the patient.

$PATH_DIR: $SRC_DIR/healtheid-notification-adapter-interface/src/main

$PATH_JDIR : $PATH_DIR/java/eu/europa/ec/healtheid/

Package: eu.europa.ec.healtheid.notification.adapter.model

Class: HealtheidNotification

File: $PATH_JDIR/notification/adapter/model/HealtheidNotification.java

Attribute	**Data type**	**Meaning**
title	String	The object of the message, e.g. the email's object.
message	String	The notification message, e.g. the body of the email.
email	String	The email to send the message.
sms	String	The phone number to send the message.
notificationSent	Boolean	The result of the sending process: `true` if the process is completed successfully, `false` otherwise.

Table B.9. Content of the HealtheidNotification Java object defined in the HeID Notification Adapter component.

B.3 HeID eIDAS HProxy: source code, resources and business logic

The eIDAS HProxy component is based on version 2.2.0 of eIDAS sources in order to use OpenSAML v3, instead of the deprecated v2 of eIDAS 1.4.3. Even though the component is bundled as a separate component, this allows its coexistence with other components using the same version of this library. The eIDAS HProxy sub-components are based on version 1.4.3 of eIDAS, thus compatible and in line with the configurations from such version. The eIDAS HProxy acts as eIDAS SP, so its implementation is based on eIDAS SP. In the following are reported the class of the eIDAS SP reference sources that are modified to make the component suitable for working in the HeID scenario.

Variable	**Value**	**Meaning**
$SRC_DIR	sources folder	Directory containing the components of the code developed in the thesis project.
$EU_EIDAS_DIR	$SRC_DIR/eidas-hproxy/src/main	Source directory of the HeID eIDAS HProxy component.
$EU_EIDAS_JDIR	$EU_EIDAS_DIR/java/eu/eidas/sp/	Java source directory of the eIDAS Demo-SP adapted to the HeID project.
$EU_HEID_JDIR	$EU_EIDAS_DIR/java/eu/europa/ec/healtheid	Java source directory of the HeID services for the eIDAS HProxy component.

Table B.10. Environment variables for understanding the programmer's manual.

B.3.1 Class: EidasMappingService

This class implements a mapping service used by the *eIDAS HProxy* component to map the encounterID with the SAML ID in order to correctly associate the SAML Request/Response with the specific encounter session. This service is realised by a `ConcurrentHashMap`, a Java collection optimised for concurrency, performance and scalability. The mapping is representet by the `<key,value> pair, where the \lstinlinekey`! is value is the SAML ID and the `value` the encounterID. The class provides *getter* and *setter* methods to access at the `Map` data structure.

Attribute: encounterEidasMap

Package: eu.europa.ec.healtheid.notification.adapter.service

Class: EidasMappingService

File: $EU_HEID_JDIR/eidashproxy/services/EidasMappingService.java:11

```
private ConcurrentHashMap <String,String> encounterEidasMap = new
    ConcurrentHashMap <> ();
```

B.3.2 Class: AuthRequest

This class is responsible to create the eIDAS authentication request and contact the eIDAS Connector. The authentication process is triggered by the Workflow Manager, after the patient clicks on link, contacting the endpoint `https: < hostname:port >/eidas-hproxy/AuthRequest` which activates the following method:

Method: startauth

Package: eu.eidas.sp

Class: AuthRequest

File: $EU EIDAS DIR/java/eu/eidas/sp/AuthRequest.java:116

```
public ModelAndView startauth(@RequestParam(value="token")String jwtToken)
```

This method creates the eIDAS SAML authentication request. It receives the encounter ID as parameter (`token`) and from which it reads the Country Code so that the component knows to which country forward the eIDAS authentication request. In addition, before starting the authentication process and after having generated the SAML message, this method maps in the `EidasMappingService` the encounterID with the SAML ID: in this way, when the SAML response comes back, the `eidas-hproxy` component is able to link the received attributes with the encounter session and communicate the result to Workflow Manager.

B.3.3 Class: AuthResponse

This class validates the eIDAS Assertion coming from the eIDAS Connector, received at the end point `https: < hostname:port >/eidas-hproxy/AuthResponse` then extracts the eIDAS attributes and sends them to the Workflow Manager.

Method: response

Package: eu.eidas.sp

Class: AuthResponse

File: $EU EIDAS DIR/java/eu/eidas/sp/AuthResponse.java:62

```
public void response(ModelMap model, @RequestParam String SAMLResponse,
    HttpServletRequest request);
```

This method accepts the eIDAS Assertion coming from the eIDAS Connector. It reads the SAML InResponseTo in order to retrieve the encounter ID (which identifies the patient who is authenticated) from the `EidasMappingService` and sends a message to the Workflow Manager with the attributes obtained from the Assertion and the encounterID.

Bibliography

[1] Schengen Area, https://ec.europa.eu/home-affairs/what-we-do/policies/borders-and-visas/schengen_en

[2] Shaping the Digital Single Market, https://ec.europa.eu/digital-single-market/en/policies/shaping-digital-single-market

[3] Creating a digital society, https://ec.europa.eu/digital-single-market/en/policies/creating-digital-society

[4] eHealth : Digital health and care overview, https://ec.europa.eu/health/ehealth/overview_en

[5] European Parliament, Council of the European Union, "Directive 2011/24/EU of the European Parliament and of the Council of 9 March 2011 on the application of patients? rights in cross-border healthcare", March 2011, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32011L0024

[6] CEF eHealth Digital Service Infrastructure, https://ec.europa.eu/cefdigital/wiki/display/EHOPERATIONS/eHDSI+Mission

[7] National Institute of Standards and Technology, https://csrc.nist.gov

[8] National Institute of Standards and Technology, An Introduction to Computer Security: The NIST Handbook, Special Publication 800-12, October 1995, DOI 10.6028/NIST.SP.800-12r1

[9] R. Shirey, "Internet Security Glossary, Version 2", RFC-7519, August 2007,

[10] William Stallings, "Computer Security Concepts" in the book "Cryptography and Network Security - Principles and Practices - Seventh Edition", Pearson, 2017, pp. 29-32

[11] Standard OASIS SAML, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[12] S.Cantor, J.Kemp, R.Philpott, E.Maler *et al.*, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard, 15 March 2005 https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[13] S.Cantor, J.Kemp, R.Philpott, E.Maler *et al.*, "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard, 15 March 2005 https://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf

[14] Rainer Hörbe, "SAML V2.0 Metadata Guide", https://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf

[15] S.Cantor, J.Kemp, R.Philpott, E.Maler *et al.*, "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard, 15 March 2005 https://www.oasis-open.org/committees/download.php/51890/SAML%20MD%20simplified%20overview.pdf

[16] T.Dierks, C.Allen, "The TLS Protocol Version 1.0" RFC-2246, January 1999

[17] William Stallings, "Transport-Level Security" in the book "Cryptography and Network Security - Principles and Practices - Seventh Edition", Pearson, 2017, pp. 549-566

[18] The SSL/TLS Handshake: an Overview, https://www.ssl.com/article/ssl-tls-handshake-overview/

[19] IBM Knowledge Center, "An overview of the SSL or TLS handshake", https://www.ssl.com/article/ssl-tls-handshake-overview/

[20] OpenSSL, cryptography and SSL/TLS toolkit, https://www.openssl.org

[21] E.Rescorla, "HTTP Over TLS?, RFC-2818, Maggio 2000

[22] European Parliament, Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons

with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)", April 2016, https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

[23] European Parliament, Council of the European Union, "Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC", July 2014, http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv%3AOJ.L_.2014.257.01.0073.01.ENG

[24] European Parliament, Council of the European Union, "Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures", December 1999, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:31999L0093

[25] Official Journal of the European Union, "The Treaty on the Functioning of the European Union", https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:12012E/TXT

[26] Marco Dogliani, Ilenia Massa Pinto, "Elementi di Diritto Costituzionale", Giappichelli Editore, 2015

[27] European Commission, Commission Implementing Regulation (EU) 2015/806 of 22 May 2015 laying down specifications relating to the form of the EU trust mark for qualified trust services https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32015R0806

[28] Agenzia per l'Italia Digitale, https://www.agid.gov.it/it/agenzia/chi-siamo

[29] STORK - Take your e-identity with you, everywhere in the EU, https://ec.europa.eu/digital-single-market/en/content/stork-take-your-e-identity-you-everywhere-eu

[30] STORK eID infrastructure as an enabler of cross-border efficiencies when interacting with public and private sectors, https://ec.europa.eu/digital-single-market/en/news/stork-eid-infrastructure-enabler-cross-border-efficiencies-when-interacting-public-and-pr

[31] eIDAS eID Profile, Technical Specifications, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eIDAS+eID+Profile

[32] Competitiveness and Innovation Framework Programme, http://ec.europa.eu/cip

[33] eHealth DSI Technical Community - OpenNCP Community Home, https://ec.europa.eu/cefdigital/wiki/display/EHNCP/OpenNCP+Community+Home

[34] 5th meeting of the eHealth Network: Agenda, Topics and Documents (Athens, 13 May 2014), https://ec.europa.eu/health/ehealth/events/ev_20140513_en

[35] Connecting Europe Facility https://ec.europa.eu/inea/en/connecting-europe-facility

[36] 2019 CEF Telecom Call - eHealth (CEF-TC-2019-2), https://ec.europa.eu/inea/en/connecting-europe-facility/cef-telecom/apply-funding/2019-ehealth

[37] DG SANTE, CEF eHealth DSI, 2019, "eHealth DSI System Architecture Specification", https://ec.europa.eu/cefdigital/wiki/display/EHOPERATIONS/System+Architecture+Specification

[38] DG SANTE, CEF eHealth DSI, 2019 "eHealth DSI Section I - Security Policies", https://ec.europa.eu/cefdigital/wiki/display/EHOPERATIONS/Section+I+-+Security+Policies

[39] eHealth Network, "Guideline on an Organisational Framework for eHealth National Contact Point", https://ec.europa.eu/health/sites/health/files/ehealth/docs/ev_20151123_co01_en.pdf

[40] DG SANTE, CEF eHealth DSI, 2019, "Identity Management Specification", https://ec.europa.eu/cefdigital/wiki/display/EHOPERATIONS/Identity+Management+Specification

[41] DG SANTE, CEF eHealth DSI, 2019, "Identity Management Specification", https://ec.europa.eu/cefdigital/wiki/display/EHOPERATIONS/Section+II+-+Security+Services

[42] A. Campanile, R. Cover, D.Davis, J. Durand *et al.*, "OASIS Standard: Basic Security Profile Version 1.1", October 2014, https://docs.oasis-open.org/ws-brsp/BasicSecurityProfile/v1.1/BasicSecurityProfile-v1.1.pdf

[43] Paolo Smiraglia, Marco de Benedictis, Andrea S. Atzeni, Antonio Lioy, Massimiliano Pucciarelli, "The FICEP Infrastructure - How We Deployed the Italian eIDAS Node in the Cloud." DOI 10.1007/978-3-319-71117-1_14

[44] M. Jones, J. Bradley, N. Sakimura, "JSON Web Token (JWT)", RFC-7519, May 2015,

[45] M. Jones, J. Bradley, N. Sakimura, "JSON Web Signature (JWS)", RFC-7519, May 2015,

[46] M. Jones, J. Hildebrand, "JSON Web Encryption (JWE)", RFC-7519, May 2015,

[47] Academic Dictionaries and Encyclopedias, "Universally Unique Identifier", https://enacademic.com/dic.nsf/enwiki/126251

[48] eHDSI MS Expert Community Home, Patient Information Notices (PIN), https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=74094942

[49] Java Programming Language, https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html

[50] The Java Virtual Machine Specification, https://docs.oracle.com/javase/specs/jvms/se8/html/index.html

[51] The Java Tutorials. Object-Oriented Programming Concepts, https://docs.oracle.com/javase/tutorial/java/concepts/

[52] Maven, https://maven.apache.org

[53] Maven Repository, The OpenSAML library, https://maven-repository.com/artifact/org.opensaml/opensaml/2.5.3

[54] Apache Tomcat, https://tomcat.apache.org

[55] MySQL, https://www.mysql.com

[56] MySQL 5.7 Reference Manual, Chapter 27, Connectors and APIs https://dev.mysql.com/doc/refman/5.7/en/connectors-apis.html

[57] Java JDBC API, https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

[58] Git, https://git-scm.com

[59] Bitbucket, https://bitbucket.org

[60] Slack, https://slack.com/intl/en-it/

[61] Spring, https://spring.io

[62] Spring MVC, https://spring.io/projects/spring-security

[63] Spring Security, https://www.thymeleaf.org

[64] Thymeleaf, https://spring.io/projects/spring-security

[65] Java Persistence API, https://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[66] Spring Data, https://spring.io/projects/spring-data

[67] Spring Email, https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-email.html

[68] Spring Boot Maven Plugin, https://docs.spring.io/spring-boot/docs/current/maven-plugin/

[69] Bootstrap, https://getbootstrap.com

[70] N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC-2045, November 1996,

[71] CEF Digital OpenNCP Bitbucket, https://ec.europa.eu/cefdigital/code/projects/EHNCP/repos/health-eid

[72] Java Platform, Standard Edition Installation Guide, JDK Installation for Linux Platforms, https://docs.oracle.com/javase/8/docs/technotes/guides/install/linux_jdk.html

[73] CEF eID Services, eIDAS Node Integration Package, eIDAS-Node version 2.2, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eIDAS-Node+version+2.2

[74] OpenNCP Community Home, OpenNCP Installation Overview, OpenNCP Installation Manual, https://ec.europa.eu/cefdigital/wiki/display/EHNCP/OpenNCP+Installation+Manual