HEARTFAID

D28 – Integration and Interoperability middleware prototype

Submission date: 19/03/2008
Due date of document: 31/01/2008

HEARTFAID

A KNOWLEDGE BASED PLATFORM OF SERVICES FOR SUPPORTING MEDICAL-CLINICAL MANAGEMENT OF THE HEART FAILURE WITHIN THE ELDERLY POPULATION

Project summary	
Project acronym:	HEARTFAID
Project identifier:	IST – 2005 – 027107
Duration of the Project:	01/02/2006 – 31/01/2009
Project Co-ordinator:	UNICAL University of Calabria (Italy)
Thematic Priority:	Information Society Technology
Instrument:	Specific Targeted Research or Innovation Project

Consortium

- UNICAL- Università della Calabria (Italy)
- UNICZ- Università degli studi Magna Graecia di Catanzaro (Italy)
- UNIMIB- Università degli studi di Milano Bicocca (Italy)
- JUMC- Jagiellonian University Medical College (Poland)
- VMWS- Virtual Medical World Solutions Ltd (United Kingdom)
- FORTHNET S. A.- Hellenic Telecommunications and Telematic Applications Company S. A. (Greece)
- SYNAP- Synapsis s.r.l. (Italy)
- CNR- Consiglio Nazionale delle Ricerche (Italy)
- FORTH-Foundation for Research and Technology Hellas (Greece)
- RBI- Rudjer Boskovic Institute (Croatia)
- AUXOL- Istituto Auxologico Italiano (Italy)

D28 – Integration and Interoperability middleware prototype

Document summary	
Document title:	D28 – Integration and Interoperability middleware prototype
Document classification:	Report/Prototype
Dissemination level:	PU
Submission date:	March 19th, 2008
Due date:	January 31st, 2008
Authors:	VMWS FORTHNET SYNAP CNR FORTH
Work package:	WP3 – Middleware, Interoperability and Integration
Report version:	1.0

Change record		
Version number	Changes	Release date
0.1	First draft of the document	10/01/2008
0.5	Advanced draft	15/02/2008
1.0	Final version	19/03/2008

Table of contents

Executive Summary

This document describes the results obtained in the context of “WP3 – Middleware, Interoperability and Integration” of the HEARTFAID project. In particular, it will describe the architecture and the implementation details on the HEARTFAID Middleware, which is responsible for the data exchange among the different modules of the HEARTFAID Platform and for guaranteeing the interoperability both inside the platform and with the external end-user world, certifying, as well, that all the incoming, outgoing and exchanging information are compliant with the data representation and communication standards.

This document follows what has been investigated, designed, and described in Deliverable “D20 – Clinical standards and first middleware prototype” (delivered at month M18) that was mainly focused on the clinical standards for data representation and on the lower layer of the middleware itself, called Integration Middleware. The technologies investigated, and the preliminary prototypes experimented, have been refined and finalised, thus achieving the main prototype that represents the core supporting architecture of the entire HEARTFAID Platform.

In order to achieve the main goal to guarantee the interoperability within the platform of services, common and well-known communication standards like Bluetooth, TCP/IP and HTTP have been adopted for implementing the interconnections between the medical devices and the Heartfaid systems and among the different software agents of the system. These have been designed in such a way to guarantee scalability and flexibility because of the large number of heterogeneous actors involved in the overall implementation of the platform.

The architectural and the technological approach is based on state of the art ideas, solutions and patterns of Enterprise Integration such as *Message Oriented Middleware*, *Enterprise Portals*, *Service Oriented Architectures* and *Enterprise Services Buses.*

Moreover, standard IHE profiles have been implemented to make the platform more flexible to the integration of new modules that are compliant with this reference standard. In particular, it is relevant to mention that the actor MPI (Master Patient Index) for the unique identification of the patients was implemented. Other profiles (that will be certified at the forthcoming IHE Connect-a-thon, to be held in Oxford on April 2008) implemented are: the Patient Demographic Query HL7 V3 (PDQ), the Patient Identifier Cross-Reference HL7 V3 (PIX), the Cross-Enterprise Clinical Documents Share (XDS), the Cross-Enterprise Sharing of Medical Summaries (XDS-MS) and the Audit Trail and Node Authentication (ATNA).

In addition to the technical implementation details, an accurate description is given about the functionalities realised, the use-case and the scenarios supported and the communication protocols specifically designed and adopted

among the components of the HEARTFAID platform. The main functionalities implemented are:

- User authentication and profiling
- Global patient enrolment and consistent identification
- Search and selection of a patient context
- AmI enrolment
- Alarm System enrolment
- Alarm handling after a critical situation has been reported by the DSS
- Reporting of clinical, observational and demographic information
- Handling of imaging data.

1. Glossary of terms

TERM	DEFINITION
AmI	Ambient Intelligence
AmIE	AmI enrolment
ASE	Alarm System enrolment
EPR	Electronic Patient Record
ESB	Enterprise Service Bus
GPE	Global Patient Enrolment
HF	HEARTFAID
HFP	HEARTFAID Platform
SOA	Service Oriented Architecture
SPC	Select Patient Context

2. Introduction: role of the Middleware in HEARTFAID

The HEARTFAID platform will act as an informative and decision support infrastructure, on the basis of which efficient and effective health care delivery organization and management models and new clinical approaches could be potentially defined and developed. The HEARTFAID platform will be an effective distributed and heterogeneous infrastructure and its core will be implemented with a multi-functional middleware layer. This layer will be, in fact, responsible for the data exchange among the different platform modules and for guaranteeing the interoperability both inside the platform and with the external end-user world. In addition, this layer should certify that all the incoming, outgoing and exchanging information are compliant with the data representation and communication standards, as well as that all the communications performed among the internal modules of the platform and between the platform and the external applications are compliant with these standards.

The global technological objective of the project is to integrate all the components using an integration and interoperability middleware, to provide a innovative platform of services to make more effective the medical-clinical management of heart diseases within elderly population.

In other words, the middleware level is the component of the HFP responsible to guarantee the efficient access and exchange of all the patients' clinical data acquired both during home monitoring (raw data) and during his/her contacts with the health facilities (complex information).

The problem is more complex than simply adopting standard languages to communicate among different applications. Institutions working on these issues are addressing also the problem of defining standard roles and workflows in order to describe accurately typical interactions inside healthcare structures and across different facilities. It is needed a totally interoperating and integrated infrastructure, able to exchange clinical data, to access distributed and shared repositories of both raw data and complex information, as well as to access computational resources and medical expertise, etc. In HEARTFAID a Middleware has been designed and developed in such a way as to guarantee the interoperability among the involved systems.

As shown in the DoW of the project, the Middleware level is logically divided in two main layers: the Integration Middleware and the Interoperability Middleware. These two layers represent two different abstraction levels of the functionalities of the Middleware.

The Integration layer will use an information-oriented integration approach to communicate and exchange data with all sources. Moreover the building of a middleware at data level will act as an open system, where existing external applications or systems like electronic health records, electrocardiographs, holter devices, etc. and also new data sources systems available in future could be easily integrated providing adapters or gateways modules, based on common standards in healthcare informative systems.

2.1 Integration Middleware

The main component of the Integration Middleware is the Data Management System that is responsible to guarantee the following features:

- All the data flowing within the entire platform are compliant with the standards identified in Task 3.2
- Management of the heterogeneous repositories and, therefore, support the management of raw data, structured information (EPR, data entry services, and so on), multimedia/other data (reports, images, ultrasound signals, and so on).

The Data Management System will use an information-oriented integration approach to communicate and exchange data with all sources. The building of this middleware at data level will act as an open system, where existing external applications or systems like electronic health records, electrocardiographs, holter devices, etc. and also new data sources systems available in future could be easily integrated providing adapters or gateways modules, based on common standards in healthcare informative systems.

In the initial idea presented in the DoW of the project, the Integration middleware was equipped with a main repository that should act as centralised data storage. Furthermore, adequate approaches based on “data replication” should have been investigated to guarantee a better data availability and to improve the performance of the analysis and processing activities.

During the analysis of the existing technologies, the initial idea was released for the following main reasons:

- It is unrealistic, both from quantitative and efficiency points of view, thinking that all the clinical data can be stored in a single repository, disregarding the high heterogeneity of these data: measurements, reports, CDSS responses, patient records, observational data and so on.
- The owners of the data (e.g. the healthcare structures for the patient records, or the service provider for the alarm contact details) typically follow their own policies for the management of these data, for the security and privacy aspects, for the accessibility to the data by the internal personnel. All these policies, which are reasonably different from institution to institution, should be accurately replicated when the data are managed by the HFP. Even though that the different policies are replicable within a single data management module, we incur in (at least) four additional problems:
    - The owners of the data will probably not trust the policies implemented in the HFP;

  - If a problem with data management occurs, the owner of the data will not receive the same technical support they would receive with the internal technical personnel;
  - Changing the policies becomes very difficult;
  - The responsibility for data loss/damage cannot be ascertained the same way as with the internal personnel;
- Last, but probably the most important, the owners of data are usually not available to provide freely their data.

Finally, we had to take into considerations adequate technologies that could support the approach of having multiple distributed sources of data. Therefore we focused on data federation solutions (i.e. the integration of a set of database schema offering an unified view of data by means of a virtual database, that maps calling and requests to the physical databases) and interface processing solutions (i.e. information importing by means of specific API) defined by external packaged applications.

In this context, the Data Management System will have to interact with all the available repositories while every module integrated into the platform will be responsible for the management of its own data. In this way, no replication of data is needed and, in addition, different specific policies for data access can be defined with a fine grain at the level of each single module.

2.2 Interoperability Middleware

This layer has the main objective to guarantee the interoperability of the HEARTFAID system components and, using the approach of service-oriented integration, to integrate the functionalities that the HEARTFAID platform will provide to the end users.

Every platform component will interact with other components, by offering and using a set of functional services. The composition methods and communication strategies between components is the main responsibility of the Interoperability Middleware. For example, the Data Management System could offer a <get-data-matrix/> useful to Knowledge Discovery component for data analysis, or this last could send the event <new-rule-validated-by-expert-available/>, useful to the Knowledge Base to store the new rule. And again, the data management system, when storing some data in own repository, could ask to middleware <please-verify-those-data/>, and the DSS connected will be activated to verify, using, if necessary, other platform components containing ontologies. Or vice versa, the DSS could be activated by the user to process the ECG signal of a patient, and send to the middleware a data request <give-me-patient-data>, that the Data Management System will satisfy.

Soon after the preliminary prototyping experiments performed with this layer, we have realised that, although a logical distinction between the integration and the interoperability middleware is necessary, the most recently developed technologies based on SOA (Service Oriented Architectures) and ESB (Enterprise Service Bus), which have been considered the most adequate technologies for the purposes of the HF project, are capable to support the functionalities of both levels. In other words, we still keep separate the functionalities that the integration and the interoperability middleware should guarantee, however from a technological point of view they have been realised by sing the same infrastructure (that will be described in more details in Section 3 - Final implementation of the HFP of services)

2.3 Structure of the document

The deliverable has been organised as follows.

In Section 2 we will describe the role of the middleware in the HEARTFAID project as well as the distinction between Integration and Interoperability Middleware. This section will show the assumption that have been released since the beginning of the project and innovative solutions adopted.

Section 3 provides and architectural description of the prototype implemented. In particular, we will describe the technologies implemented and the methodological approaches adopted.

Section 4 gives an overview of the functionalities implemented to achieve the major goals of the project. For each functionality we will provide the use-case scenario, the activity diagram of the software modules involved and the communication protocol implemented.

Finally, Section 5 will describe the advantages of using the standard IHE profiles and the specific profiles that have been adopted.

3. Final implementation of the HFP of services

3.1 The ESB

In a scenario where the data sources are several, distributed, pre-existing and heterogeneous, it is not possible to think of a solution that creates a new data source by physically merging the others. Moreover it is not practicable to implement the integration in a point-to-point fashion or even adopting a simple Message oriented Middleware (MoM).

The key issues was to choose the correct ways of implementing the integration of the systems taking into account flexibility of the system to be integrated, its legal access policy and the technologies it is implemented with and that it can support.

Enterprise Service Bus (ESB): is a new approach to integration that can be used as the foundation for a loosely coupled highly distributed and thus highly scalable integration network. An ESB is a standards based integration platform that combines messaging, web services, data transformation and intelligent routing to reliably connect and coordinate the interaction of a significant numbers of diverse applications across extended enterprises with transactional integrity.

The main goals achieved by approaching the integration of Heartfaid application modules by utilizing an ESB can be summarized in:

- **Non intrusive**: the integration is transparent and seamless related to its impact on the application design and implementation of the single HF modules.
- **Pervasive:** it is able to span an extended domain that crosses technologies and organizational boundaries such as (Firewalls, Internet, Departmental access policies, etc.)
- **Standards based**
- **Highly distributed and event-driven**
- **Autonomous but federated**: cooperation as a means for granting performances, robustness and fault tolerance
- **Secure and reliable**
- **Remotely configurable**: natively distributed architectures can be controlled by remote management consoles to reduce maintaining costs.
- **Incrementally adoptable**: every module was integrated at a proper timeline.

All Heartfaid devices communicate among each other by exchanging messages on the ESB. Since the ESB offers also services of message routing and transformation it removes the burden of implementing these very important functionality from the application designer making him or her concentrate on the logic of the proper AmI Device he's implementing.

The routing and transformation mechanisms implemented by the ESB are highly configurable also remotely. Moreover a big variety of different protocols including their secure implementations has been used to face different heterogeneous entities that are able, in this way, to automatically and dynamically connect to the ESB without requiring necessarily a proxy software.

As related to the resource management and storage system the development team considered the adoption of three different storage methods:

- A traditional, SQL based, relational database (PostgreSQL) for storage of raw observational data and for the backend of all transactional tools that will be used to implement the ESB
- A XML:DB based native XML database (eXist) for storing XML or text based documents (Configurations, scripts, presentation markup)
- A traditional file system for storing log files.

Resources such as configurations, scripts, queries, presentation sheets, transformation files and so on are uniformly identified by Universal Resource Names (URNs). URNs are a specialization of the concept of URIs where the URI actually identifies the name of the resource instead of the location like in URLs.

Proper URN resolver modules take care of mapping URNs to the actual resources, fetching the resource and applying configurable caching policies.

In a first analysis step the different modules that contribute to the Heartfaid applications have been identified. The main application modules are developed by the Heartfaid partners and range from full blown applications with their own native interfaces and storages, through head- and state-less web services to code-on-demand libraries for very specific computations.

- The **AmI** is an Ambient Intelligence framework that handles the storage and monitoring of observational data acquired by sensors and interacts with environmental actuators. It has been developed by SYNAP in conjunction with FORTHNET and VMWS who delivered the sensing infrastructure.
- The **eCRF** module is a web based Electronic Patient Record (EPR) that offers native user interfaces for input and edit patient related demographic and clinical information. It has been developed by JUMC.
- The **groupSMS** is an application for sending short messages on mobile phones with handling of address lists and accounting. It is developed by FORTHNET.
- The **CDSS** is the code of a Clinical Decision Support System developed by CNR in conjunction with RBI who designed the clinical ontology of the system.
- The **HFP** is an image analysis and archiving toolkit used at CNR for implementing algorithms for the analysis of clinical images.

The scenario is depicted in the following picture showing the application with their datamodel and each one with its applicative core and native access interface.

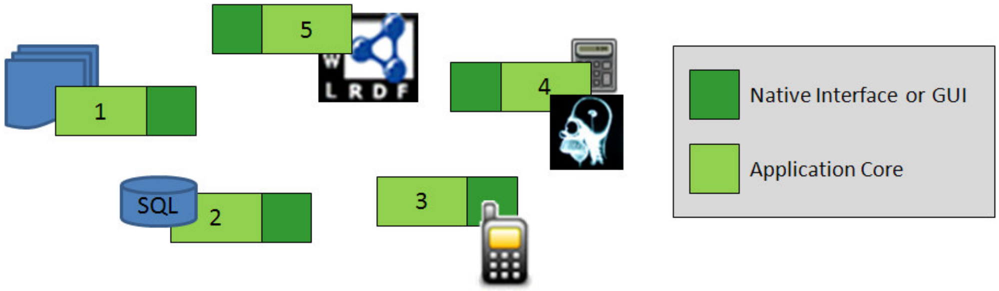


Figure 1: Datamodel of the involved applications

This short list already presents several issues in terms of integration. Every module is written in its own language (Java, VB, HTML, C/C++, ontology modeling languages ...), according to its own application model (web application, distributed application, legacy stand alone rich application, code-on-demand library ...), requiring its own data type (name-value strings, XML, relational data) and exposing native interfaces in its own presentation formalism (HTML view, Rich Internet Application, Java Swing, OS native GUIs).

Moreover every module has its own way of implementing interconnection and networking (from nothing at all through bare TCP/IP to web services and JMS based messaging) and for handling life-cycle management.

In such a complex scenario it was immediately clear that an integration strategy could not be based on point-to-point agreements among each pair of application modules that have to be integrated. This approach would lead to an explosive situation from a maintenance point of view and to a worst case approach from a scalability point of view.

Finally there were also difference and mismatches in how the different modules handle common or correlated information such as for example patient identification. Nonetheless they all contribute with important functionality and even more important data.

Thus it has been decided to adopt a state of the art approach based on the installation of an ESB.

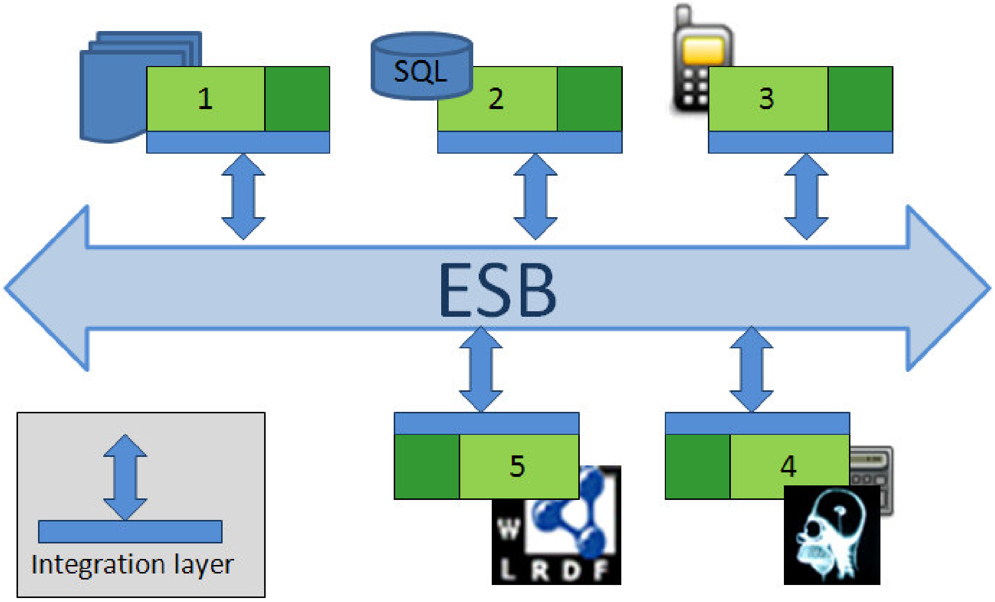


Figure 2: Architectural viewpoint

In Figure 2 a possible scenario is depicted from an architectural viewpoint.

An adapter has been written for plugging the application modules to the ESB. The wrapped applications are now able to loosely interconnect and communicate with the other modules through a message passing infrastructure that is the foundation of the ESB.

At this point the integration architect took care of designing and configuring the interactions among the application modules, the data transformations, quality of service, synchronization, workflow specification, life-cycle management and so on for granting the implementation of the services that want to be exposed at global Heartfaid level. This activities were performed leveraging the technologies and tool support contributed by the underlying ESB thus dramatically reducing the development effort.

So far the integration has been analyzed only from a technical viewpoint. There were also requirements on functionality that had to be added for completing the integration middleware as a whole.

In particular the four main functionalities that have been taken into account were:

- **Uniform access to shared resources:** besides the data and information locally generated by each application module, a system like the integrated Heartfaid platform requires a lot of information to be shared. This information has to put into the middleware in a way that is as transparent and easy as possible for the application itself without however impacting to heavily on the retrieval costs.A best practice is to use a combination of two pattern of application integration [4]: a ***documental repository*** and a ***registry service.*** The documental repository stores data in a weakly structured way that allows the developer to access it with very simple primitives such as *put* and *get* calls thus removing from the software engineer the burden of writing complex queries in native database idioms. The registry service allows any module of a system to retrieve a *reference to* a piece of information stored in the system (possibly a repository) through querying by a subset of the meta-data identifying it.

- **Unique identification of patients:** it is very important that the possibly different identifications of the patient that are kept by the different application modules are uniformed and synchronized across the whole Heartfaid information system. This could not be done by enforcing a common identifier to all the participant application modules because of the high cost it would have to change such a typically hard coded and structural information. Thus a proper entity called ***Master Patient Index*** (MPI) has been integrated. It implements the task of synchronizing different patient identification schemes as transparently as possible.

- **Generation of visual information:** a proper ***Display*** has been implemented for generating reports of information that is retrieved through a federated query on the different systems. Reports can be generated at specific time instants or on demand. They can be stored in a documental repository or sent through standard protocols (like email) to registered consumers.

- **User access control and interaction:** most of the services exposed at the overall Hearfaid level have to be interactively started or guided by human operators. This implies that a proper strategy of integration has been thought of also at the presentation level. According to best practices and patterns in enterprise application integration the most suitable solution is to setup an ***Enterprise Web Portal*** (shortly Portal). The Portal is in charge of unifying the access to the Heartfaid system for all kind of purposes (configuration, service interaction, dissemination). Unifying the access means also controlling the access policies and access rights of the different classes of users that will be able to selectively access the Heartfaid system according to profiling information. Moreover the Portal is also a mean of realizing an early prototype of a ***Single Sign On*** (SSO) system that grants the user access to all the available components, data or functionality logging in only once.

The following picture shows the above mentioned architecture enriched by the described application modules.

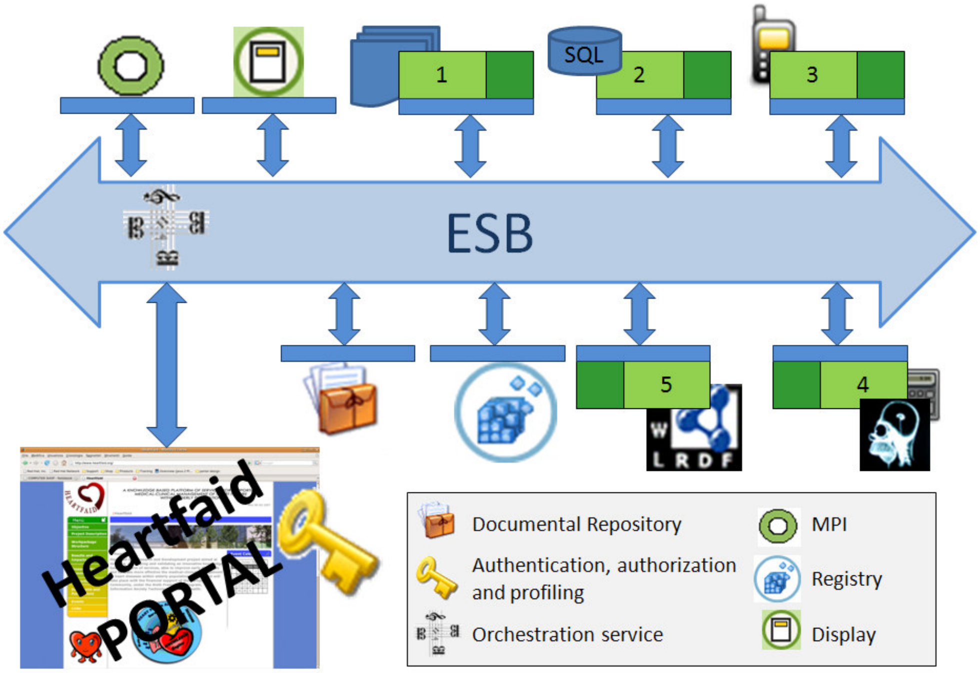


Figure 3: The Heartfaid architecture

As can be seen the middleware also makes use of one of the most important feature of ESBs: the Orchestration Service (OS). The OS is highly configurable, scriptable actor of the ESB which is involved in the workflow management regulating the interactions and data exchanges of the different application modules participating in the integrated Heartfaid platform.

3.2 Implementation details

3.2.1 General considerations

The development team adopted open source technologies whenever possible and to leverage on agile software design methodologies for achieving rapid and experience prototyping.

A significant shift from coding and programming based solution towards configuration based, extensible development patterns such as Inversion of control and dependency injection has been adopted. Scripting languages and declarative programming approaches have been adopted.

As regards other technological guidelines the implementation of the Heartfaid integration middleware consists in:

- Portable programming environment

- Lightweight programming model (workflow based) for keeping the code of the adapters configurable and scalable.
- A reliable, asynchronous message passing infrastructure.
- Easy to use reporting tool
- XML based information exchange, transformation and filtering
- Native XML database for quickly storing and querying XML documents

There are several technologies that address the above mentioned implementation aspects or subsets of them. The followin is the list of the adopted ones.

Portable programming environments are used to programming and deploy applications, modules, libraries without having to worry too much about the target platform. The choice of implementation will be Java because of the huge amount of software solutions that already exist for nearly every available platform.

As **Lightweight programming model** a workflow engine called MIDA has been used. MIDA has been written by SYNAP and could be fit to the requirements of the Heartfaid project thanks to its generality. Actually MIDA is an executor of graph based workflows where each node represents an atomic operation such as for instance: an interaction with a db, an XML transformation or a communication. The graph representing workflows are expressed in a very simple XML document and thus can be reconfigured easily to adapt to emerging requirements.

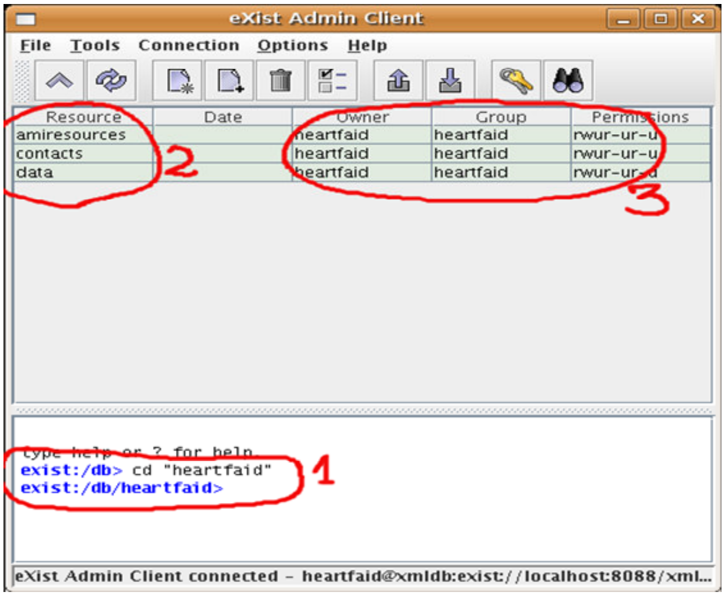
OpenJMS has been choosen as **asynchronous and reliable messaging service**. Actually there was no need to adopt it since all communications that occure in the implemented scenarios could be re-factored to use the ESB standard communication patterns thus reducing the overhead in terms of maintenance and configuration of the middleware installation.

For the sake of simplicity a very simple and general purpose XML based protocol has been developed for implementing the communications among the HF modules that communicate through **XML messaging** (see Section 4). This messages are used to communicate with the application modules that were slightly more flexible and could offer a REST based application programming interface (AmI, Portal, eCRF, MPI and DISPLAY). Anyway proprietary protocols have been used were the uniformity of messaging could not be enforced (cDSS, GroupSMS).

Jboss [18] is a Java J2EE and JSR168[16] based portal product which is open source. This is the choice for implementing the HF Portal since it is also a J2EE application server implementing all J2EE specification including a built-in JMS implementation (JbossMQ).

As an **easy to use reporting tool** the development team used iReport [22] that grants a very usable interface for designing the look of a report document. Data can be fed into the report by executing queries on XML documents or relational databases. The runtime of iReport is called Jasper report which has been integrated with Mida to be used for automatic report generation at specific time intervals.

For implementing the storing of **XML documents** a natively XML database has been choosen. The choice was made selecting **eXist**. eXist is the most complete opensource and freely available implementation of the XML:DB specification featuring an enormous amount of extra functionality such as: access through *WEBDAV*, *REST* and *XMLRPC* protocols; embeddable XSL transformation, support for Xupdate language and a lot of very powerful extension modules. It is also very simple to install on different platforms and to configure. In the heartfaid middleware it is used intensively for implementing the repository of patient related data (demographic, observational) and configuration. Actually its wide use allows the development team to avoid integrate another (relational) persistence thus again reducing the cost of maintenance. In the following picture some screen shots of the database are shown representing the structure of Heartfaid data.

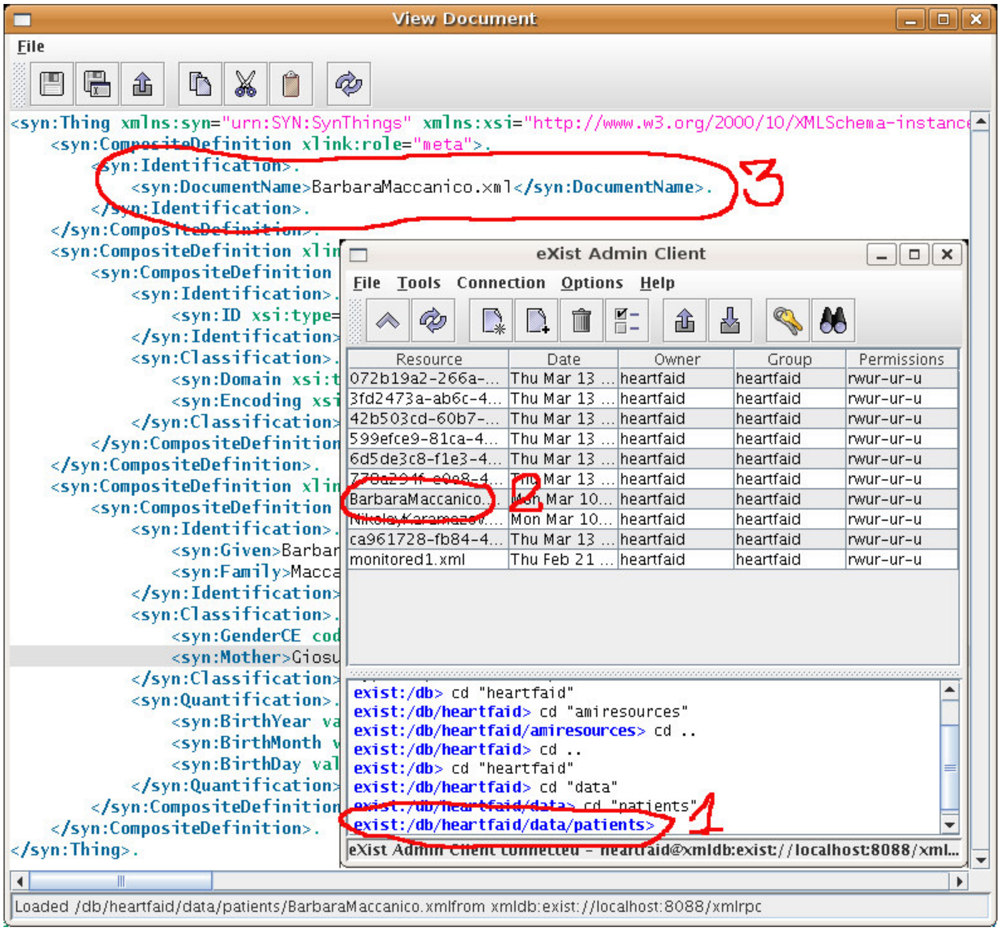


Resource	Date	Owner	Group	Permissions
amiresources		heartfaid	heartfaid	rwur-ur-u
contacts		heartfaid	heartfaid	rwur-ur-u
data		heartfaid	heartfaid	rwur-ur-u

```
type help or ? for help.
exist:/db> cd "heartfaid"
exist:/db/heartfaid>
```

eXist Admin Client connected – heartfaid@xmldb:exist://localhost:8088/xml...

Figure 4: 1) the path to the collection containing the Heartfaid data. 2) the subcollection containing the configuration for AmI, the contacts for emails and sms and patient data. 3) eXist supports authentication for accessing critical data



View Document

```
<syn:Thing xmlns:syn="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance
    <syn:CompositeDefinition xlink:role="meta">.
        <syn:Identification>.
            <syn:DocumentName>BarbaraMaccanico.xml</syn:DocumentName>.
        </syn:Identification>.
    </syn:CompositeDefinition>.
    <syn:CompositeDefinition xlin
        <syn:CompositeDefinition
            <syn:Identification>.
                <syn:ID xsi:type=
            </syn:Identification>
            <syn:Classification>.
                <syn:Domain xsi:t
                <syn:Encoding xsi
            </syn:Classification>
        </syn:CompositeDefinition
    </syn:CompositeDefinition>.
    <syn:CompositeDefinition xlin
        <syn:CompositeDefinition
            <syn:Identification>.
                <syn:Given>Barbar
                <syn:Family>Macca
            </syn:Identification>
            <syn:Classification>.
                <syn:GenderCE cod
                <syn:Mother>Giosu
            </syn:Classification>
            <syn:Quantification>.
                <syn:BirthYear va
                <syn:BirthMonth v
                <syn:BirthDay val
            </syn:Quantification>
        </syn:CompositeDefinition
    </syn:CompositeDefinition>.
</syn:Thing>.
```

eXist Admin Client

Resource	Date	Owner	Group	Permissions
072b19a2-266a-...	Thu Mar 13 ...	heartfaid	heartfaid	rwur-ur-u
3fd2473a-ab6c-4...	Thu Mar 13 ...	heartfaid	heartfaid	rwur-ur-u
42b503cd-60b7-...	Thu Mar 13 ...	heartfaid	heartfaid	rwur-ur-u
599efce9-81ca-4...	Thu Mar 13 ...	heartfaid	heartfaid	rwur-ur-u
6d5de3c8-f1e3-4...	Thu Mar 13 ...	heartfaid	heartfaid	rwur-ur-u
778a294f-e0e8-4...	Thu Mar 13 ...	heartfaid	heartfaid	rwur-ur-u
BarbaraMaccanico...	Mon Mar 10...	heartfaid	heartfaid	rwur-ur-u
NikolayKaramazov....	Mon Mar 10...	heartfaid	heartfaid	rwur-ur-u
ca961728-fb84-4...	Thu Mar 13 ...	heartfaid	heartfaid	rwur-ur-u
monitored1.xml	Thu Feb 21 ...	heartfaid	heartfaid	rwur-ur-u

```
exist:/db> cd "heartfaid"
exist:/db/heartfaid> cd "amiresources"
exist:/db/heartfaid/amiresources> cd ..
exist:/db/heartfaid> cd ..
exist:/db> cd "heartfaid"
exist:/db/heartfaid> cd "data"
exist:/db/heartfaid/data> cd "patients"
exist:/db/heartfaid/data/patients>
```

eXist Admin Client connected - heartfaid@xmldb:exist://localhost:8088/xml...

Loaded /db/heartfaid/data/patients/BarbaraMaccanico.xmlfrom xmldb:exist://localhost:8088/xmlrpc

Figure 5:** **1)** **Path to the collection containing the patient demographic data.** **2)** **An example XML document for a patient.** **3)** **eXist has a built-in editor for editing XML resources. Here we highlight the meta-data section of a document.

Finally the most important choice to be made was the one related to the ESB implementation. Mule ESB has been choosen because it is the most widespread, completely opensource and freely available solution for ESBs [5].

Mule has been integrated with MIDA to deliver a very powerful combination of tools that allowed the development team to quickly deliver templates for the implemented information flows and proceed in an agile way to re-factor the templates without having to consider big coding overheads.

Mule functionalities used are:

1. HTTP, TCP and SMTP transport protocols used in synchronous and asynchronous ways

2. Quartz scheduling for time based events

3. Content based routing, transformation and filtering through Xpath and XSLT on passing by XML documents

4. Scenarios and functional design of the HFP

In this section we report the functionalities that have been implemented in the HF platform. A first version of these services were already introduced in the Deliverable “D20 - Clinical standards and first middleware prototype”, however they have been refined during the development of the functionalities, therefore it is worthwhile to describe them again with the changes implemented.

For each service we will report the use-case, which defines the high level view of the service implemented, the related activity diagram and, where it applies, the communication protocols implemented to guarantee the interoperability among the modules of the platform.

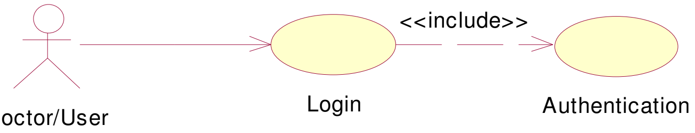
Apart from the global services, such as ‘Login’ and ‘Image Upload’ (for which it is not necessary to define a communication protocol or it has not been defined yet because not relevant/required for the general scenario), the following scenarios have been implemented, which are considered of basic importance to support the general scenario addressed by the project:

- GPE – global patient enrolment
- SPC – select patient context
- AmIE – AmI enrolment
- ASE – Alarm System enrolment

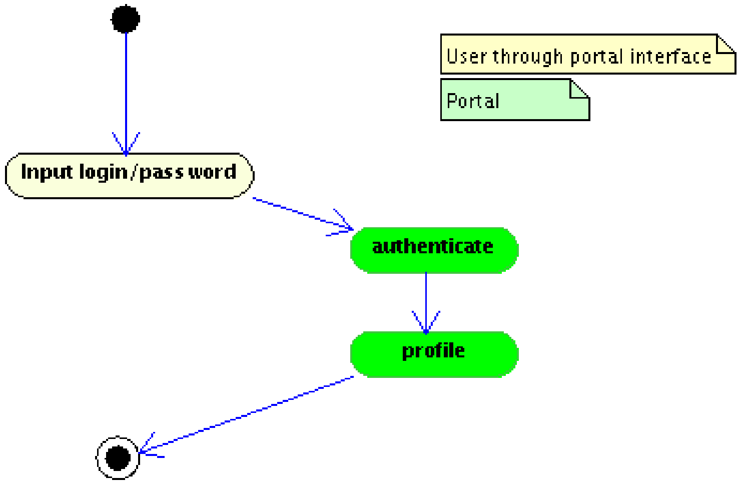
4.1 Login

4.1.1 Use-case

This use case has the goal to authenticate the user who is trying to access the platform functionalities. The provided user name and password are matched against the profiling database that is maintained directly by the Portal.
The user profile can be used both to define the access rights of the user and to automatically direct the user to a specific section and on authorization the right presentation page is shown.
Prerequisite: the user accesses the public section of the Portal from where he/she can login. The Portal user profiling database contains a record describing the user's profile.
Result: the user is prompted with a page in the Portal which states the success or failure of the login procedure.
Involved modules: the Login is performed as a whole by the Portal thus no other application module is involved. No messages are exchanged with the platform.



4.1.2 Activity-diagram



4.2 Set Patient Context (SPC)

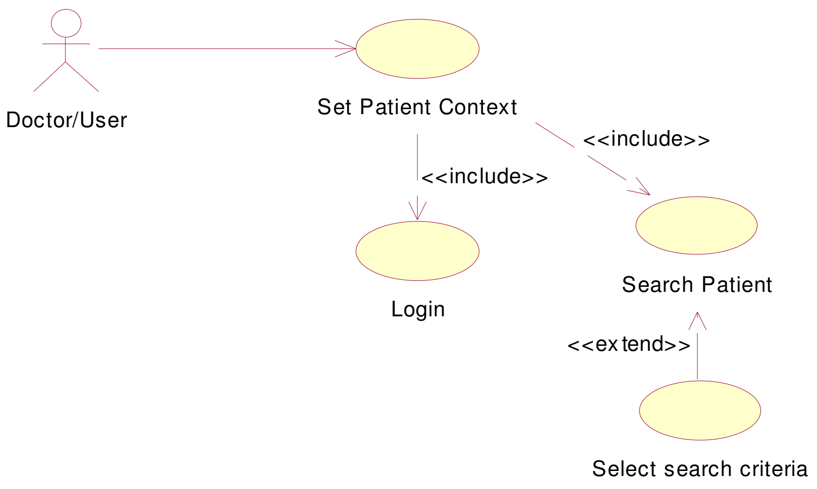
4.2.1 Use-case

SPC is a basic use-case that sets the user (e.g. the doctor) in a proper patient context. This means that the user handles information related to a specific patient until he/she closes the application or switches to another patient. The patient has to be identified through a proper patient search interface in the Portal. The identification of the patient occurs through the input of patient demographic information and after the search process completes, one patient has to be selected from the list of possibly matching patients.

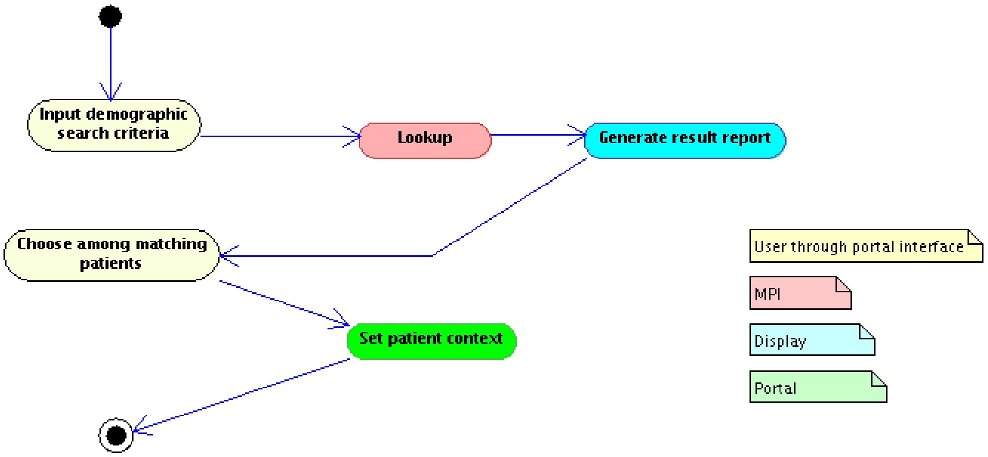
Prerequisite: a *Login* has to be executed successfully.

Result: the user is prompted with a page containing a list of patients matching the query and he/she has to choose one among the matching patients. At this point the patient context is set.

Involved modules: besides the Portal, the Master Patient Index is involved for executing the demographic query and the Display for generating a report of all matching patients.



4.2.2 Activity-diagram


Input demographic search criteria
Lookup
Generate result report
Choose among matching patients
Set patient context
User through portal interface
MPI
Display
Portal

4.2.3 Communication protocol

In the SPC scenario the Portal sends a request message composed of the patient corroborating identity data. The Heartfaid platform returns a message containing the identity information of all matching patients. The portal can then set the context choosing one out of the matching patients.

An example is the message sent by the portal for searching a patient by its corroborating identity information. In this message the portal searches for all patients whose name starts with *Vito* and whose surname starts with *Gat*. Any other field is considered neutral to the search operation (i.e. a request with all the fields set to the empty string results in all patients known by the platform).

The platform must return not more than fifty records of matching patients and the result-set must be ordered ascending according to the sort criteria which for this request is the patient's family name.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd">
  <action>patient.search</action>
  <payload>
        <record>
                <parameter name="patient.id"></parameter>
                <parameter name="patient.given">Vito</parameter>
                <parameter name="patient.family">Gat</parameter>
                <parameter name="patient.birthday"></parameter>
                <parameter name="patient.birthmonth"></parameter>
                <parameter name="patient.birthyear"></parameter>
                <parameter name="patient.gender"></parameter>
        </record>
        <howmany>50</howmany>
        <order> ASC</order>
  </payload>
</request>
```

The HF platform will respond with a proper response message that will contain a set of matching records. The *howmany* field reports the actual count of records returned. The portal can set it's context to one of the matching patients or re-execute the query with different criteria.

Note that the message contains an AA code in the ack element and the refaction attribute reproduces the query code.

The gender parameter will be encoded in MFU notation (M=male, F=female, U=unspecified).

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd">
  <ack refaction="patient.search">AA</ack>
  <payload>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="patient.name">Vito</parameter>
                <parameter name="patient.surname">Gattuso</parameter>
                <parameter name="patient.birthday">12</parameter>
                <parameter name="patient.birthmonth">3</parameter>
                <parameter name="patient.birthyear">1970</parameter>
                <parameter name="patient.gender">M</parameter>
        </record>
        <howmany>1</howmany>
  </payload>
</response>
```

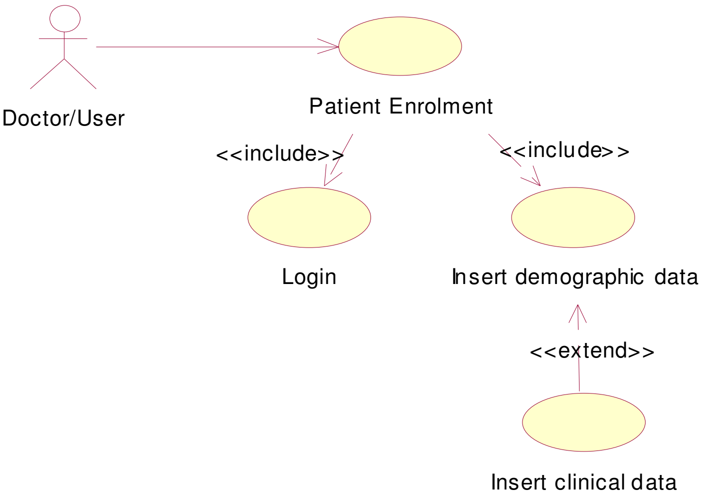
4.3 Global Patient Enrolment (GPE)

4.3.1 Use-case

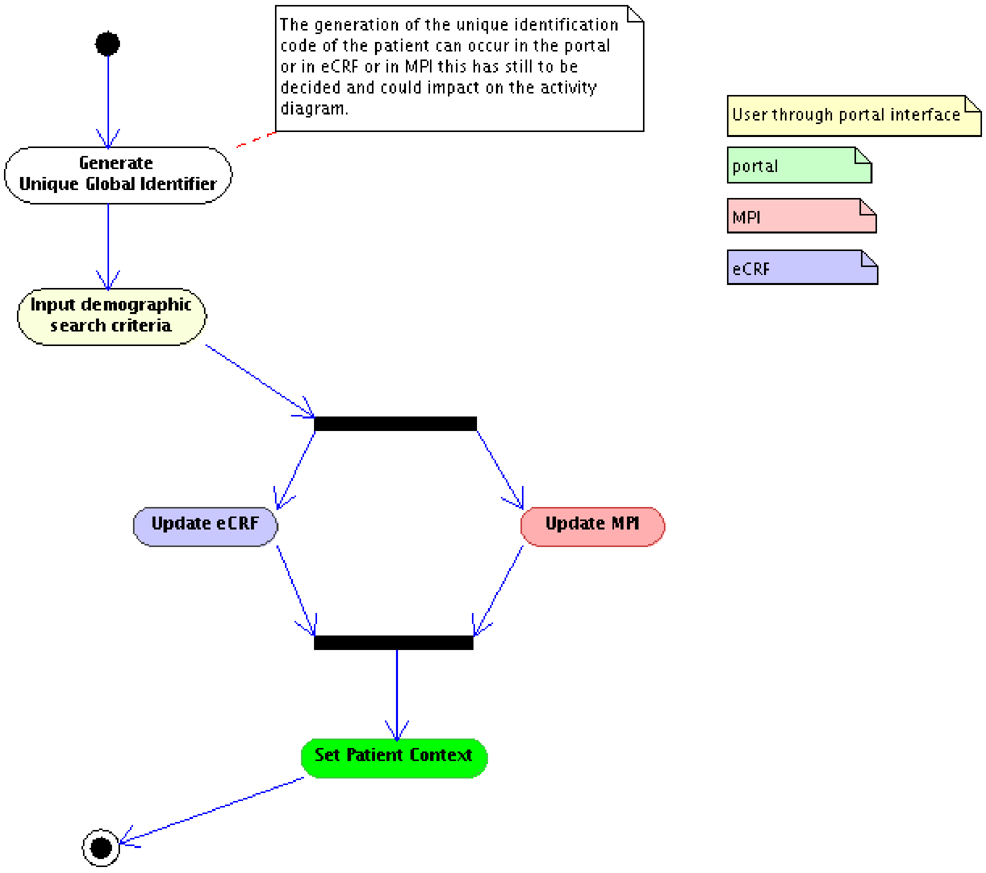
The *Global Patient Enrollment* describes the action of *inserting* a new patient into the global Heartfaid platform. This operation is different from *local enrollment procedures* that actually register patient related data into the different application modules possibly by using native interface functionality. With the *GPE* the platform and in particular the MPI and the eCRF become aware of the presence of:

- a new Heartfaid Patient identity
- the patient's demographic data
- the patient's inclusion/exclusion criteria

Prerequisite: the user requesting a *GPE* has to be logged and authorized by the Portal.
Result: after a *GPE* the MPI and eCRF are synchronized with the demographic information of the new patient. The Portal context is set to the enrolled patient.
Involved modules: the GPE is guided by a proper interface in the Portal. The new inserted data (identification and demographic data of a patient) are sent to the MPI and the eCRF.



4.3.2 Activity-diagram


The generation of the unique identification code of the patient can occur in the portal or in eCRF or in MPI this has still to be decided and could impact on the activity diagram.
Generate Unique Global Identifier
Input demographic search criteria
Update eCRF
Update MPI
Set Patient Context
User through portal interface
portal
MPI
eCRF

4.3.3 Communication protocol

It is possible from the portal to enrol a patient into the HF platform. For this purpose the portal must send an XML message as follows.

Please note that all fields are required (according to the HL7 standard specification).

The patient's birth-date is split in its basic components in order to avoid formatting problems. The gender attribute should be encoded in MFU format.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
   <action>patient.enrol</action>
   <payload>
            <record>
                        <parameter name="patient.given">Vito</parameter>
                        <parameter name="patient.family">Gattuso</parameter>
                        <parameter name="patient.mother">Lorusso</parameter>
                        <parameter name="patient.birthday">12</parameter>
                        <parameter name="patient.birthmonth">1</parameter>
                        <parameter name="patient.birthyear">1970</parameter>
                        <parameter name="patient.gender">M</parameter>
            </record>
        </payload>
</request>
```

If the enrolment procedure completes correctly the HF platform will return the following acknowledgement message.

```
<?xml version="1.0" encoding="UTF-8"?>
<response version="1.0" xmlns="urn:SYN:SynThings"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:SYN:SynThings common.xsd">
        <ack refaction="patient.enrol">AA</ack>
</response>
```

As an example the following message shows how error conditions are notified to the portal. In this case suppose one of the required enrol parameter was not sent.

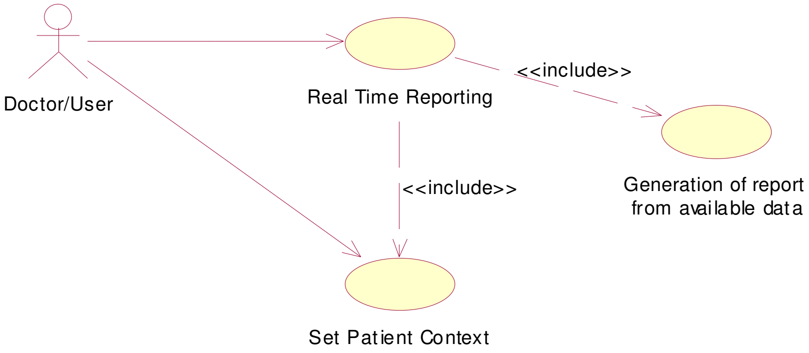
```
<?xml version="1.0" encoding="UTF-8"?>
<response version="1.0" xmlns="urn:SYN:SynThings"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:SYN:SynThings common.xsd">
        <ack refaction="patient.enrol">AE</ack>
        <errors>
                <error code="101">
                        <display>Required field missing</display>
                        <severity>E</severity>
                </error>
        </errors>
</response>
```

4.4 Real Time Reporting of patient data (RTR)

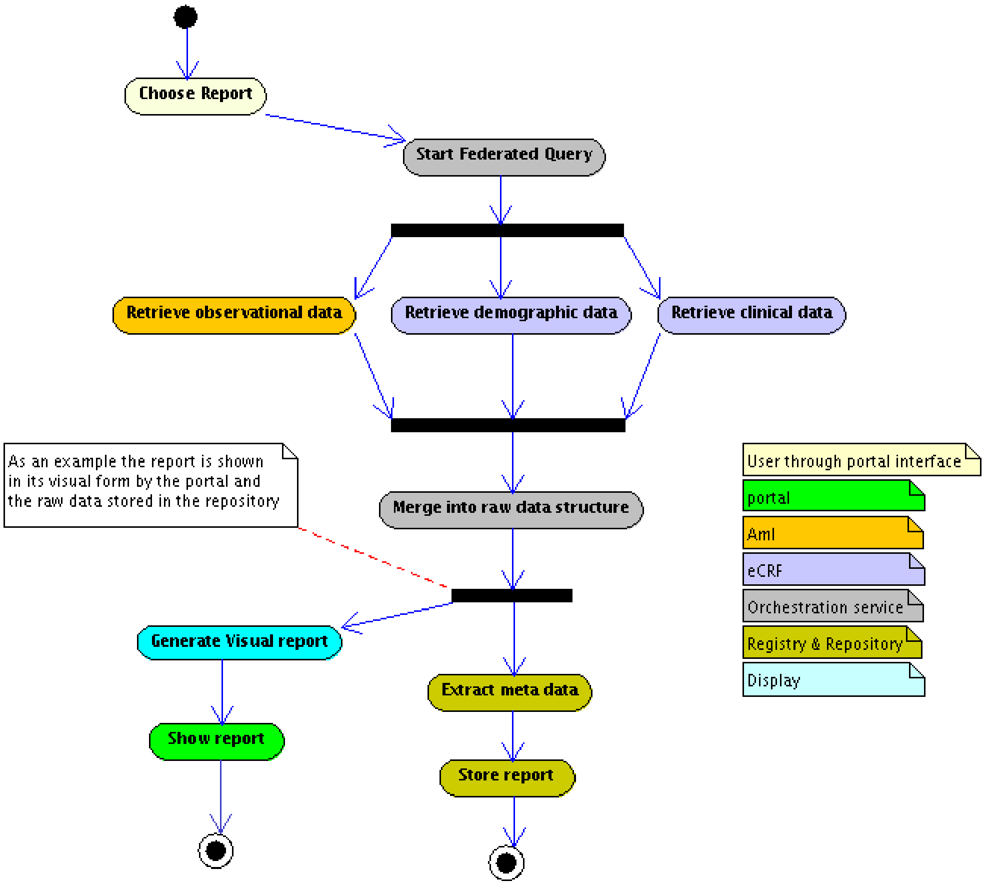
4.4.1 Use-case

The *RTR* use-case is meant to be the operation through which data that is stored locally in each application scenario is retrieved for the assembly of global reports that are then rendered on a specific displaying device by the Display module. The *RTR* matches the *Federated Query* integration pattern. The list of available reports is visible in a proper view of the Portal. The user could select the report and the format. At that point the Portal triggers the orchestration service which is responsible for steering the federated query, the gathering of the result and the production of the visual report.
Prerequisite: *Login* and *SPC* have to be executed. The Orchestration Service has to be made aware of the workflow of Federated Query.

Result: a report is sent back to the Portal. Optionally the report could also be sent to an email address or stored in the documental repository (in a possibly different format).
Involved modules: The Portal allows the user to set the patient context and choose the report and then it should be able to display it. The *Orchestration Service* steers the federated query and redirects the resulting report. The Display produces the visual report. The Registry and repository could be involved in the storage of the report data.



4.4.2 Activity-diagram


Choose Report
Start Federated Query
Retrieve observational data
Retrieve demographic data
Retrieve clinical data
Merge into raw data structure
As an example the report is shown in its visual form by the portal and the raw data stored in the repository
Generate Visual report
Show report
Extract meta data
Store report
User through portal interface
portal
Aml
eCRF
Orchestration service
Registry & Repository
Display

4.4.3 Communication protocol

The RTR (as well as the FWR described in the next paragraph), is a complex scenario. In fact, in theory it is possible to define an indefinite number of different reports using the information available on the platform. We can range from statistical reports on the measurements acquired with the remote devices, to report on the cardiologic health condition of the patient derived from the clinical data of EPR, as well as from reports of the DSS outputs to graphics obtained from home/hospital monitoring sessions, and so on. Of course, it is not purpose of the project to implement a number of reports, rather then to demonstrate the feasibility of this feature.

For this reason, considering also the importance, for the project, of both home and hospital monitoring of selected patients through the continuous acquisition of clinical measurements, we focused on the recovery of data acquired using the remote medical devices.

Moreover, since this choice is surely one of the most complex scenarios in terms of involved modules that need to communicate (typically the Portal, the ESB, the MPI, the Registry, the AmI module, the remote devices and different repositories) we will demonstrate the feasibility of this service and the flexibility of the architecture implemented.

The main goal addressed is to provide any software modules of the HFP with the possibility to send messages for executing queries on observed data. For example in the real-time reporting RTR scenario one component of the federated query could be the request for body parameters of the patient measured during the last week. Another scenario could be the display of a simple charts in the portal as a short-cut to home monitored data.

In the following example the first message represents a request sent for getting the heart rate and the pulse observations of a patient with id VG in the time interval from December, 12 2006 at 5pm to January, 12 at the same time. As stated in the request the result should be returned in descending order (for observations the sort criteria is the timestamp of the observation) and in pages of at most fifty records.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd">
  <action>ami.queryobservations</action>
  <payload>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="phenomenon.id">hr</parameter>
                <parameter name="query.fromDay">12</parameter>
                <parameter name="query.fromMonth">12</parameter>
                <parameter name="query.fromYear">2006</parameter>
                <parameter name="query.fromHour">17</parameter>
                <parameter name="query.fromMinute">00</parameter>
                <parameter name="query.toDay">12</parameter>
                <parameter name="query.toMonth">1</parameter>
                <parameter name="query.toYear">2007</parameter>
                <parameter name="query.toHour">17</parameter>
                <parameter name="query.toMinute">00</parameter>
        </record>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="phenomenon.id">pulse</parameter>
                <parameter name="query.fromDay">12</parameter>
                <parameter name="query.fromMonth">12</parameter>
                <parameter name="query.fromYear">2006</parameter>
                <parameter name="query.fromHour">17</parameter>
                <parameter name="query.fromMinute">00</parameter>
                <parameter name="query.toDay">12</parameter>
                <parameter name="query.toMonth">1</parameter>
                <parameter name="query.toYear">2007</parameter>
                <parameter name="query.toHour">17</parameter>
                <parameter name="query.toMinute">00</parameter>
        </record>
```

```
            <howmany>50</howmany>
            <order>DESC</order>
   </payload>
</request>
```

Since the resultset contains more records than what was specified in the request howmany field (i.e. fifty records), the response carries, besides the howmany field also the continuation pointer that will be used to hook up, on request, the next page of records. In cases where the returned page contains less than the requested page size the howmany field will contain the actual number of records returned and no continuation point will be present.

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version="1.0">
   <ack refaction="ami.observationquery">AA</ack>
   <payload>
            <record>
                        <parameter name="phenomenon.code">pulse</parameter>
                        <parameter name="phenomenon.description">The peripheral pulse</parameter>
                        <parameter name="phenomenon.uom">bpm</parameter>
                        <parameter name="observation.day">12</parameter>
                        <parameter name="observation.month">12</parameter>
                        <parameter name="observation.year">2006</parameter>
                        <parameter name="observation.hour">18</parameter>
                        <parameter name="observation.minute">3</parameter>
                        <parameter name="observation.second">12</parameter>
            </record>
            <record>
                        <parameter name="phenomenon.code">hr</parameter>
                        <parameter name="phenomenon.description">The heart rate</parameter>
                        <parameter name="phenomenon.uom">bpm</parameter>
                        <parameter name="observation.day">12</parameter>
                        <parameter name="observation.month">12</parameter>
                        <parameter name="observation.year">2006</parameter>
                        <parameter name="observation.hour">18</parameter>
                        <parameter name="observation.minute">8</parameter>
                        <parameter name="observation.second">10</parameter>
            </record>
            <!-- .... -->
            <!-- 48 more records -->
            <!-- .... -->
            <howmany>50</howmany>
            <continuation>uuid-1234-xdce@50</continuation>
   </payload>
</response>
```

To get the following page of records the requester must resend the same query message with the continuation pointer added to the message. The continuation pointer has simply to be copied from the previous response message.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version="1.0">
```

```
  <action>ami.queryobservations</action>
  <payload>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="phenomenon.id">hr</parameter>
                <parameter name="query.fromDay">12</parameter>
                <parameter name="query.fromMonth">12</parameter>
                <parameter name="query.fromYear">2006</parameter>
                <parameter name="query.fromHour">17</parameter>
                <parameter name="query.fromMinute">00</parameter>
                <parameter name="query.toDay">12</parameter>
                <parameter name="query.toMonth">1</parameter>
                <parameter name="query.toYear">2007</parameter>
                <parameter name="query.toHour">17</parameter>
                <parameter name="query.toMinute">00</parameter>
        </record>
        <howmany>50</howmany>
        <continuation>uuid-1234-xdce@50</continuation>
        <order>DESC</order>
  </payload>
</request>
```

Please note that missing parameters in the request mean that those parameters are transparent to the request. For example a missing

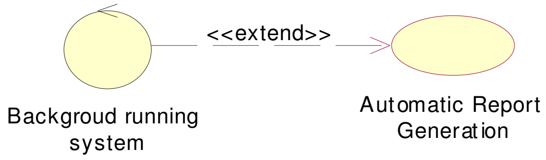
```
<parameter name="phenomenon.id">hr</parameter>
```

means that observations should be returned for all the present phenomena.

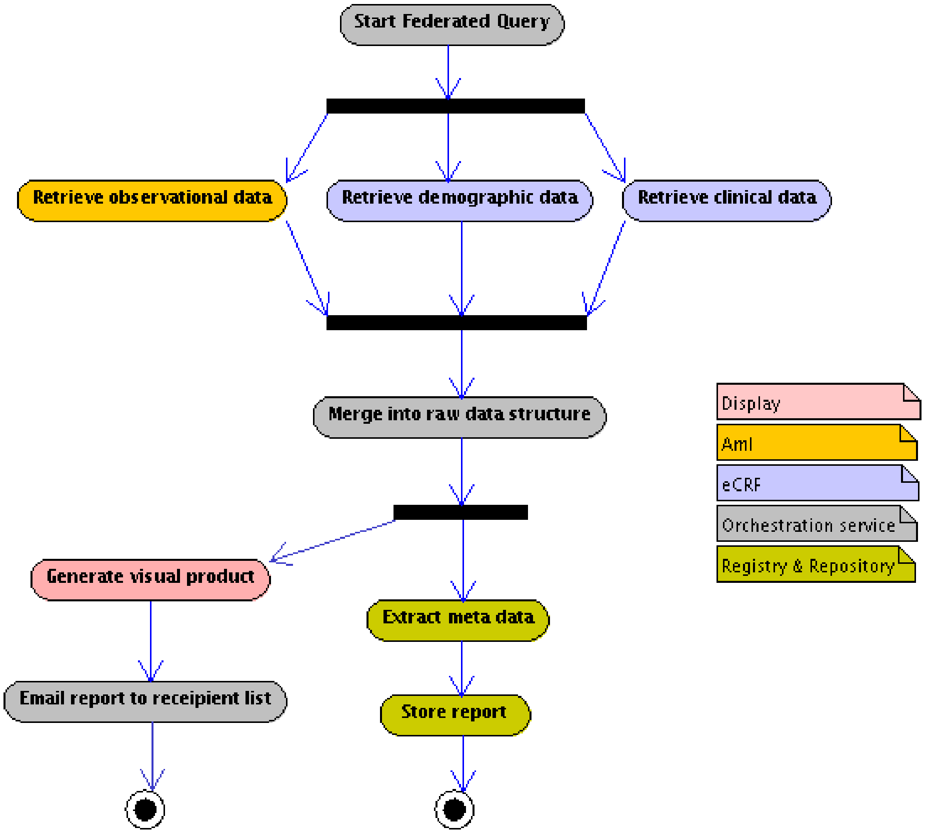
4.5 Forward Reporting of patient data (FWR)

4.5.1 Use-case

The *FWR* is different from the RTR because it matches the *Forward Cache* integration pattern. With this approach data is pushed towards the final product (a report in this case) autonomously by the Orchestration Service (on a specific event or at a specific time schedule). The data gathered is transformed into a visual product by the Display and pushed to its final destination that could be an email address or the Repository from where it can be retrieved later.
Prerequisite: the system runs autonomously the Orchestration Service has to be made aware of the workflow of Federated Query and the events that trigger it.
Result: a report that is sent to a specific destination or saved in the Repository.
Involved modules: the same modules as in RTR are involved with exception of the Portal.



4.5.2 Activity-diagram



4.5.3 Communication protocol

This scenario is similar to the previous RTR. In particular, it is a special case of the RTR protocol therefore the same protocol described in Section 4.4.3 can be used with few changes. As said above, it not purpose of the project to define the protocol schemas for all the possible reports that might be created, rather than to demonstrate the feasibility of the approach adopted.

4.6 AmI Enrolment (AmIE)

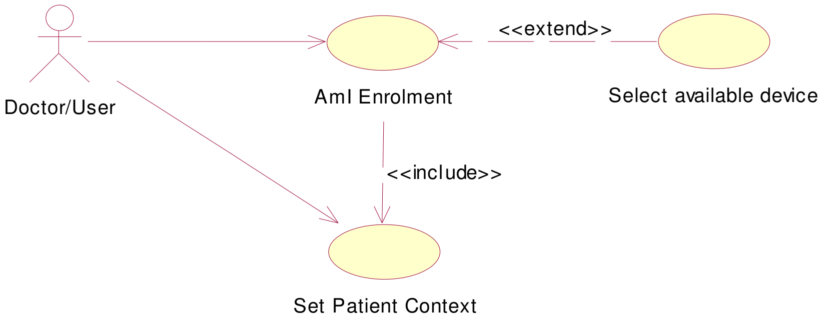
4.6.1 Use-case

The Portal is able to show a list of resources (sensors, actuators, processes) that can be assigned for home monitoring. The user can select a subset of resources to assign to the patient in the context. This assignment list is then sent to the AmI application module that will update the list of associations. It should also be possible to query the current assignments of the patient for reassignment or un-enrollment.

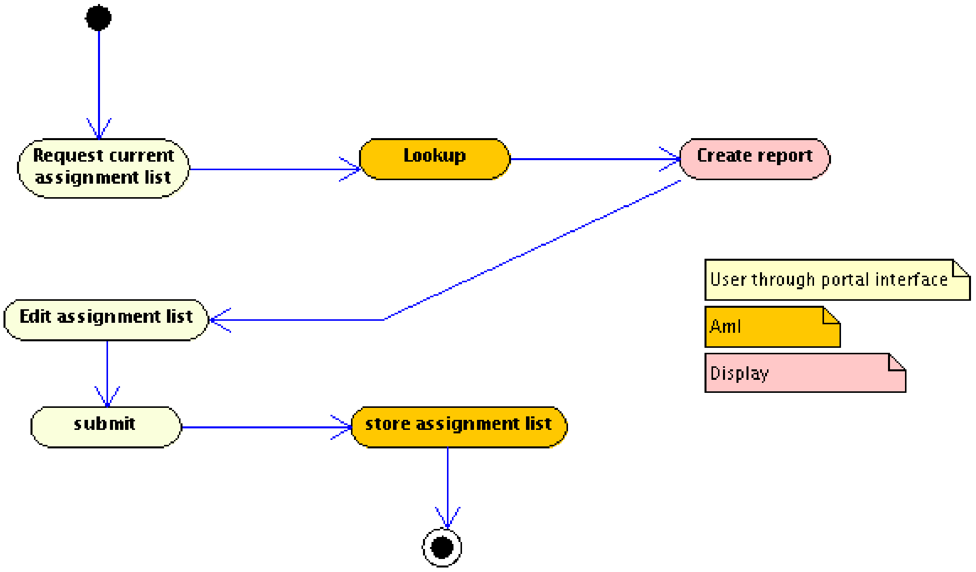
Prerequisite: Login and SPC have to be executed.

Result: a list of assignment of resources are associated with the patient inside the AmI platform.

Involved modules: Portal, AmI.



4.6.2 Activity-diagram



4.6.3 Communication protocol

The AmIE scenario is used to associate a patient with ambient intelligence resources. The flow of information is implemented as follows:

1. The portal request the current available and assigned services passing in the patient id.
2. The HF platform responds with a resultset containing the list of availbale services and the information on what services are already assigned to the patient.
3. The portal can modify the assignment status and send back the new status
4. The HF portal acks the request

The XML message that has to be generated for point one of the workflow:

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
 <action>ami.getservices</action>
  <payload>
        <record>
                <parameter name="patient.id">VG</parameter>
        </record>
  </payload>
</request>
```

The HF platform responds with the following message. In this example two services are available that monitor peripheral pulse and heart rate. The patient is already monitored (since March, 12 2007) for its peripheral pulse. But not yet for the heart rate.

Note that for each phenomenon a unique identifying code is transmitted together with the default display which is the english description of the phenomenon. If there are different procedures of measuring a phenomenon there should be a different encoding for the two.

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd">
  <ack refaction="ami.getservices">AA</ack>
  <payload>
        <record>
                <parameter name="phenomenon.code">pulse</parameter>
                <parameter name="phenomenon.display">The peripheral pulse</parameter>
                <parameter name="phenomenon.uom">BPM</parameter>
                <parameter name="phenomenon.monitored">true</parameter>
                <parameter name="phenomenon.startDay">12</parameter>
                <parameter name="phenomenon.startMonth">3</parameter>
                <parameter name="phenomenon.startYear">2007</parameter>
        </record>
        <record>
                <parameter name="phenomenon.code">hr</parameter>
                <parameter name="phenomenon.display">The heart rate</parameter>
                <parameter name="phenomenon.uom">BPM</parameter>
                <parameter name="phenomenon.monitored">false</parameter>
        </record>
        <howmany>2</howmany>
  </payload>
</response>
```

At point 3 of the workflow the portal will have to send back the new status encoding. The following message represents a template of this communication. With this example message the portal asks the HF platform to assign a heart rate monitoring resource to the patient with id VG and to keep the assignment of the pulse sensor.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
 <action>ami.enrol</action>
  <payload>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="phenomenon.id">pulse</parameter>
                <parameter name="phenomenon.monitored">true</parameter>
        </record>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="phenomenon.id">hr</parameter>
                <parameter name="phenomenon.monitored">true</parameter>
        </record>
  </payload>
</request>
```

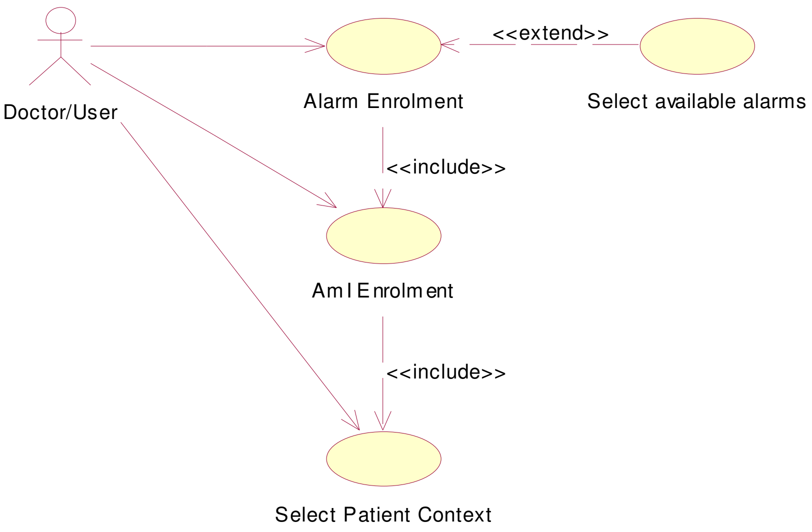
Since the operation of assigning an AmI resource to a patient is an asynchronous operation, the HF platform responds to the last request simply with an ack message.

```
<?xml version="1.0" encoding="UTF-8"?>
<response version="1.0" xmlns="urn:SYN:SynThings"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
        <ack refaction="ami.enrol">AA</ack>
</response>
```

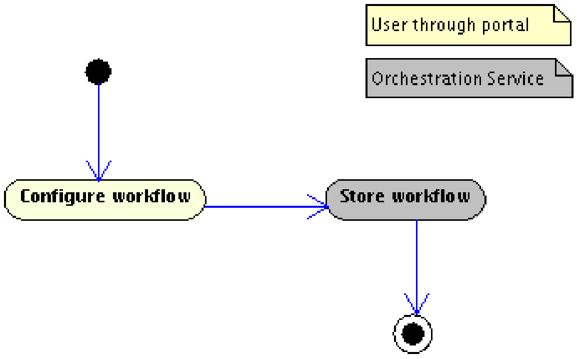
4.7 Alarm System Enrolment (ASE)

4.7.1 Use-case

The Portal is able to show a list of alarm resources that can be configured and activated. The user can select a subset of alarms to be activated for the patient in the context. Alarms typically involve different modules that play the role of value producers, decision support, reporting, storage and notification. Thus the configuration of an alarm mainly consists in the definition of workflow for the *Orchestration Service* and the triggering conditions that trigger it.
Prerequisite: Login and SPC have to be executed.
Result: a new workflow description for the *Orchestration Service*.
Involved modules: Portal, Orchestration Service.



4.7.2 Activity-diagram



4.7.3 Communication protocol

The workflow for Alarm enrolment is quite complex. Alarms are associated with particular outputs of the execution of specific knowledge base rules that reside in the cDSS module. Thus a patient has first to be enrolled into the DSS system by assigning him or her a set of DSS resources. Afterwards the patient is assigned a set of notification resources (queried from the HF platform) which represent the contact persons to be notified by means of a set of available protocols. A possible formalization of the workflow is as follows:

1. Portal queries the availble and assigned DSS services given a patient id
2. The HF platform responds with a resultset containing the list of availbale services and the information on what services are already assigned to the patient.
3. The Portal enables the modification of the assigned services and for each service the modification of the possible contact. The possible contacts can be queried from the portal through a proper search contact form.
4. The portal sends the new configuration of assigned service and notification contacts to the HF Platform
5. The platform acks.

With the following message the Portal makes the query of point 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
  <action>cdss.getservices</action>
   <payload>
          <record>
                   <parameter name="patient.id">VG</parameter>
          </record>
   </payload>
</request>
```

This is a possible response of the HF platform. The DSS observes a phenomenon which has its encoding and its display. The field phenomenon.monitored says if the patient is aready monitored by the DSS for this phenomenon.

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
   <ack refaction="cdss.getservices">AA</ack>
   <payload>
          <record>
                   <parameter name="phenomenon.code">decompensation</parameter>
                   <parameter name="phenomenon.display">The cardiac decompensation
                   </parameter>
                   <parameter name="phenomenon.monitored">false</parameter>
          </record>
```

```
        <howmany> 1</howmany>
  </payload>
</response>
```

The portal can query the notification contact persons associated with the patient by issuing the following query message to the HF platform.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
  <action>notification.getservices</action>
  <payload>
        <record>
                <parameter name="patient.id">VG</parameter>
        </record>
  </payload>
</request>
```

The platform will respond with a message that accords to the following template. The meaning of the message is that there are 4 available notification services. Two SMS service (because there are two providers for example), one mail service and a log service. For each service the *notification.assigned* field says whether the notification service is assigned (i.e. some contacts have been set) to the patient.

Every *contact.id* field represents one person to contact with the protocol encoded in the record field *notification.code.* In the example an alarm condition related to the patient wil be sent as a sms through sms provider 1 to the contacts 1023xc and 1024dd, through an email server to contact 1024dd and finally to the system logger.

```
<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
  <ack refaction="notification.getservices">AA</ack>
  <payload>
        <record>
                <parameter name="notification.code">SMS</parameter>
                <parameter name="notification.description">One SMS service</parameter>
                <parameter name="notification.protocol">sms</parameter>
                <parameter name="notification.assigned">true</parameter>
                <parameter name="contact.id">1023xc</parameter>
                <parameter name="contact.id">1024dd</parameter>
        </record>
        <record>
                <parameter name="notification.code">SMS2</parameter>
                <parameter name="notification.description">Another SMS service</parameter>
                <parameter name="notification.protocol">sms</parameter>
                <parameter name="notification.assigned">false</parameter>
        </record>
        <record>
                <parameter name="notification.code">MAIL</parameter>
                <parameter name="notification.description">One mail service</parameter>
                <parameter name="notification.protocol">mailto</parameter>
                <parameter name="notification.assigned">true</parameter>
                <parameter name="contact.id">1024dd</parameter>
        </record>
        <record>
                <parameter name="notification.code">LOG</parameter>
                <parameter name="notification.description">One log service</parameter>
                <parameter name="notification.protocol">syslog</parameter>
                <parameter name="notification.assigned">true</parameter>
        </record>
        <howmany>4</howmany>
 </payload>
</response>
```

To change the notification assignment of a patient the portal will have to send a message similar to the following template. In the example one notification through SMS is removed and the notification to the system logger is disabled.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
  <action>notification.enrol</action>
  <payload>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="notification.code">SMS</parameter>
                <parameter name="notification.assigned">true</parameter>
                <parameter name="notification.contact">1024dd</parameter>
        </record>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="notification.code">MAIL</parameter>
                <parameter name="notification.assigned">true</parameter>
                <parameter name="notification.contact">1024dd</parameter>
        </record>
        <record>
                <parameter name="patient.id">VG</parameter>
                <parameter name="notification.code">LOG</parameter>
                <parameter name="notification.assigned">false</parameter>
        </record>
  </payload>
</request>
```

Finally there are the request and reply messages that are used to retrieve personal contact information.

In the example the portal requests all contacts whose surname starts with the string “Bianchi” and two record are matched in two different hospital organizations.

```
<?xml version="1.0" encoding="UTF-8"?>
<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
  <action>contact.search</action>
  <payload>
```

```
        <record>
                <parameter name="contact.id"></parameter>
                <parameter name="contact.name"></parameter>
                <parameter name="contact.surname">Bianchi</parameter>
                <parameter name="contact.displayname"></parameter>
        </record>
        <howmany>50</howmany>
  </payload>
</request>

<?xml version="1.0" encoding="UTF-8"?>
<response xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd" version=”1.0”>
  <ack refaction="contact.search">AA</ack>
  <payload>
        <record>
                <parameter name="contact.id">Bia001</parameter>
                <parameter name="contact.name">Mario</parameter>
                <parameter name="contact.surname">Bianchi</parameter>
                <parameter name="contact.displayname">m.bianchi</parameter>
                <parameter name="contact.organization">OO RR - Bergamo</parameter>
        </record>
        <record>
                <parameter name="contact.id">Bia002</parameter>
                <parameter name="contact.name">Susanna</parameter>
                <parameter name="contact.surname">Bianchi</parameter>
                <parameter name="contact.displayname">s.bianchi</parameter>
                <parameter name="contact.organization">ASL Le 1</parameter>
        </record>
        <howmany>2</howmany>
  </payload>
</response>
```

4.8 Alarm handling (AH)

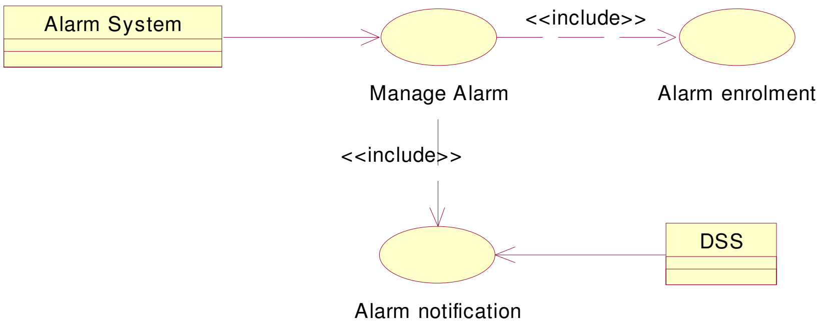
4.8.1 Use-case

The *AH* use case consists in the execution of a workflow specification that has to be configured during an *ASE* use case upon the verification of the trigger event. The *AH* is a completely autonomous use case. After gathering all the needed information the cDSS module is asked to verify the input condition against the configured ontology for checking whether an alarm has actually to be produced. If an alarming condition could be verified then the workflow provides a description of what modules have to be alerted.

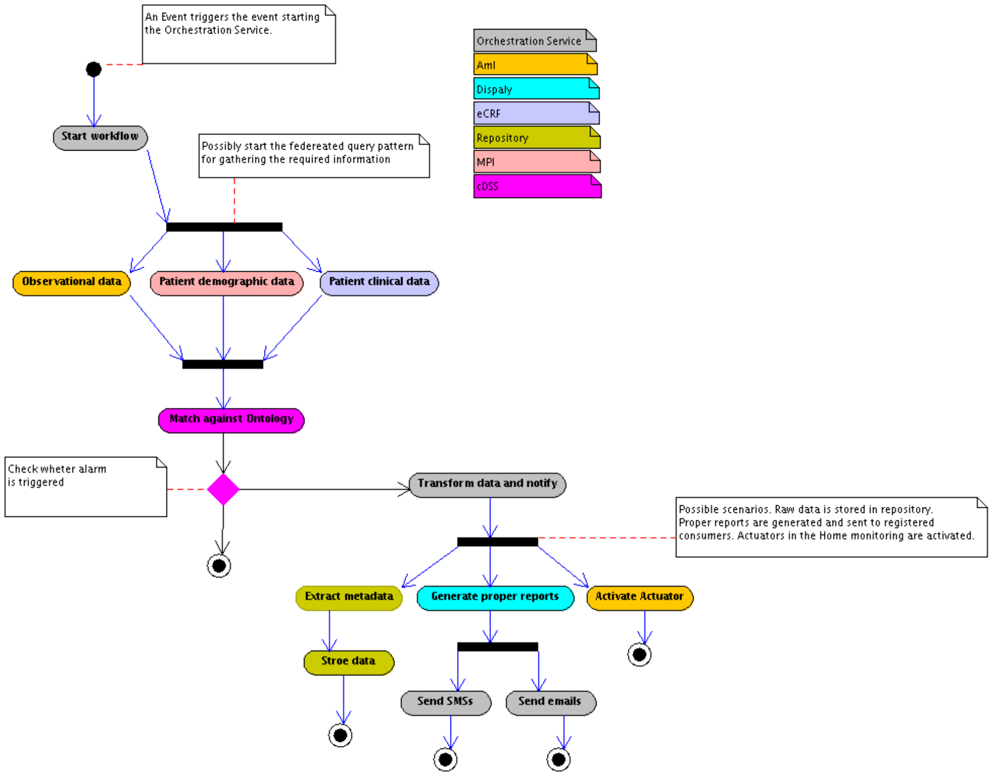
Prerequisite: an alarm has to be configured by an *ASE* use case.

Result: the result is described by the configuration of the orchestration service it could be the notification of emails or SMS to a configured participant list, a report stored in the Repository, a triggering message for an AmI actuator, a flag set in the clinical database of the eCRF ...

Involved modules: Orchestration Service, all application modules.



4.8.2 Activity-diagram


An Event triggers the event starting the Orchestration Service.
Orchestration Service
AmI
Dispaly
eCRF
Repository
MPI
cDSS
Start workflow
Possibly start the federeated query pattern for gathering the required information
Observational data
Patient demographic data
Patient clinical data
Match against Ontology
Check wheter alarm is triggered
Transform data and notify
Possible scenarios. Raw data is stored in repository. Proper reports are generated and sent to registered consumers. Actuators in the Home monitoring are activated.
Extract metadata
Generate proper reports
Activate Actuator
Stroe data
Send SMSs
Send emails

4.8.3 Communication protocol

The workflow related to the Alarm handling involves also the messages exchanged with the DSS. Once a new observation has been acquired, the ESB is notified about the new data, therefore it is able to recover the new measurements and to invoke a specific service of the DSS.

The request to the DSS can be send using the following message:

```
<?xml version="1.0" encoding="UTF-8"?>
<EarlyDetectionOfDecompensationConditions>
```

```
    <Patient MPI="A001">
        <Sex>male</Sex>
        <Age>27</Age>
        <NYHA>3</NYHA>
        <Smoke>no</Smoke>
        <Alcohol>0</Alcohol>
        <CurrentSystolicBloodPressure>140</CurrentSystolicBloodPressure>
        <CurrentHeartRate>60</CurrentHeartRate>
        <CurrentWeight>88</CurrentWeight>
        <CurrentTotalBodyWater>17.0</CurrentTotalBodyWater>
        <PreviousSystolicBloodPressure>135</PreviousSystolicBloodPressure>
        <PreviousHeartRate>65</PreviousHeartRate>
        <PreviousWeight>87</PreviousWeight>
        <PreviousTotalBodyWater>17.5</PreviousTotalBodyWater>
    </Patient>
</EarlyDetectionOfDecompensationConditions>
```

The DSS has now all the information needed to perform the computation. Once the outcome of the decision supporting algorithms is ready, the answer is send to the ESB with the following message:

```
<?xml version="1.0" encoding="UTF-8"?>
<EDDC_advice>
    <Patient MPI="A001">
        <Alert Level="2">call you doctor NOW!</Alert>
        <Suggestion accuracy="0.9051">very high risk of decompensation</Suggestion>
    </Patient>
</EDDC_advice>
```

A specific module of the ESB will process the answer of the DSS and if the alert level requires to send an SMS, a message is sent to the Alarm System:

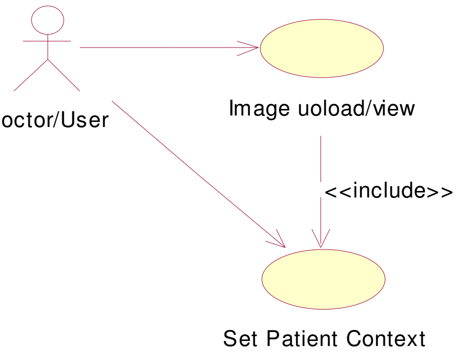
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE XML SYSTEM "http://gsms.forthnet.gr/sms.dtd">
<XML>
         <Request>SENDMESSAGES</Request>
         <Customer>
                  <Username>synapsis</Username>
                  <Password>*********</Password>
         </Customer>
         <Subject>.......</Subject>
         <Recipients>
                  <Phone>ZZZXXXYYY</Phone>
         </Recipients>
         <Message>
                  <TEXT>Patient Vito Gattuso: very high risk of decompensation</TEXT>
         </Message>
</XML>
```

Finally the Alert System forwards the message to the phone number of the final recipient (e.g. the cardiologist who has in charge Vito Gattuso) by using an SMS.

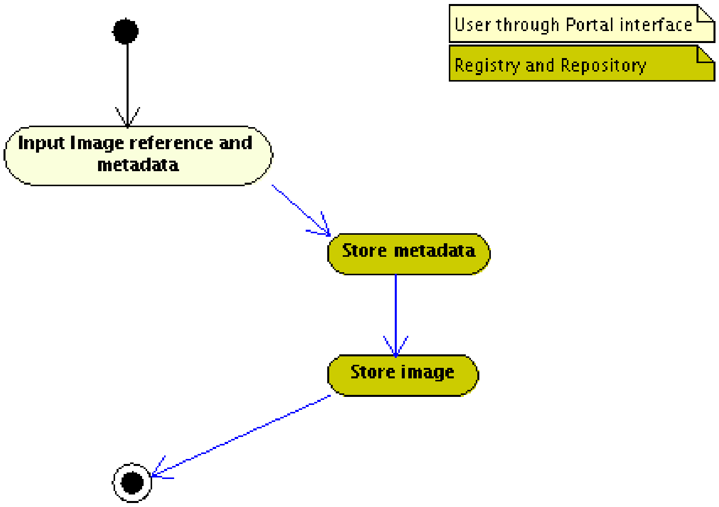
4.9 Image Upload

4.9.1 Use-case

Description: the IU use case consists in allowing an authorized and logged user to upload a DICOM image to the image repository that is maintained in the middleware.
Prerequisite: the user has to be logged and authorized.
Result: a DICOM image is stored in the Repository and its metadata are kept in the registry for retrieval.
Involved modules: Portal, Repository, Registry.



4.9.2 Activity-diagram


User through Portal interface
Registry and Repository
Input Image reference and metadata
Store metadata
Store image

5. Adopting the IHE standard

5.1 Basic principles

Health care government policy makers have to change the face of health care delivery, especially across the developed western countries, taking into account the new relationships between the health care services suppliers and the patients: the consumers are more and more demanding for high quality care, the managers are struggling to deliver health care services at reasonable costs while, on the other hand, there is an increasing diffusion of chronic diseases as well as, due to the progressive ageing of the population, a rising number of patients that need long-term continuous health assistance; clinicians are placing increasing emphasis on the practice of evidence-based medicine whereas, at the same time, citizens are improving their sensitivity regarding the clinical risk management.

To face these critical issues, the “everywhere” and “everytime” availability of the information is becoming more and more a crucial aspect of the Healthcare Information Systems (HIS). For this reason the integration of the patients data is today a priority of the HISs, which are moving their focus from the vertical management of the departmental clinical data from the horizontal management of the clinical pathways of the patients and of their continuity of care.

In this scenario, the Middleware of the HIS plays a key role because it has to guarantee the interoperability of heterogeneous legacy applications and solutions, the communication of the patient’s data and the interactions among different healthcare structures. Another key aspect is the management of the propriety, the sharing and the replication of the information: the policies of the middleware allow to reduce the replication of the information, to guarantee the security and the confidentiality of the data.

For face all these critical aspects the middleware components has to provide two key features:

1. to identify univocally the patients, both across the different care environments end within the commercial ICT solutions available in the medical domain and
2. to adopt integration and interoperability standards, in order to guarantee the use, the sharing and the exploitation of all the personal, clinical, medical and wellbeing information related to the patient, acquired both during his/her life, i.e. during a long period of time, and at different locations/healthcare structures.

The Heartfaid Middleware focalized in these aspects integrating the actor MPI (Master Patient Index) for the unique identification of the patients and building the infrastructure of the HF platform following the key integration profiles suggested by IHE (Integrating Healthcare Enterprise) for the management of the shared information of the patients.

5.2 Profiles implemented

For the first prototype of the middleware, the consortium implemented these integration profiles using a XML based communication protocol, because the HF platform integrates, in this prototype, modules of the different partners of the consortium (eCRF, AmI platform, DSS etc).
Anyway the used XML protocol includes all the necessary information of an HL7 message and the implementation follows the transactions suggested by IHE. To completely adhere to the IHE integration profile Synapsis will participate to the next European IHE Connect-a-thon that will take place in Oxford, from the 7th April to the 11th April 2008 with the objective to obtain the conformity to the integration profiles described below.

5.2.1 Patient Demographic Query HL7 V3 (PDQ) and Patient Identifier Cross-Reference HL7 V3 (PIX)

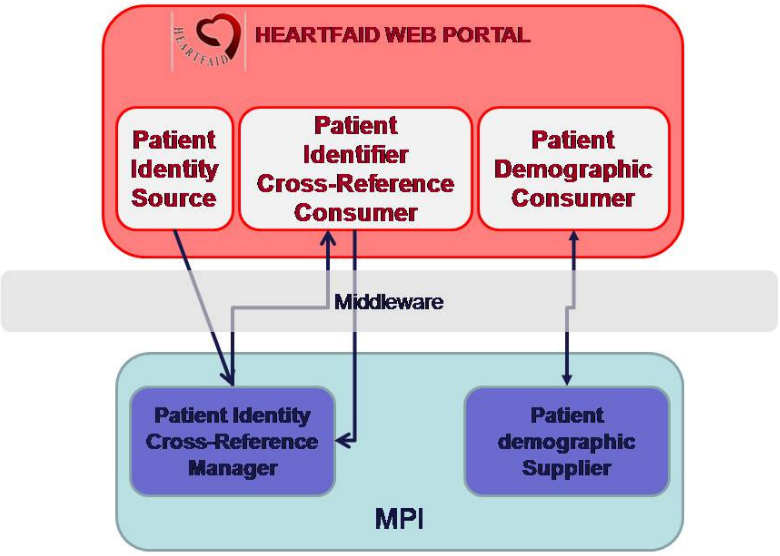
Description
These two profiles allow to manage the demographic information of the patients and to assure their unique identification. In particular, the **PDQ** profile allows multiple distributed applications to query a central patient information server for a list of patients, based on user-defined search criteria. Patient demographics data can be entered directly into the application from which the user is querying by picking the appropriate record from a list of possible matches (called here a patient pick list).
On the other hand, the PIX profile supports the Cross-referencing of patient identifiers from multiple Patient Identification Domains. These Cross-referenced patient identifiers can then be used by "identity consumer" systems to correlate information about a single patient from sources that "know" the patient by different identifiers. This allows a Clinician to have more complete view of the patient information.

Actors
The PDQ profile involves the actor *Patient Demographic Consumer*, that allows a user to associate information with a patient at the point of care and the actor *Patient Demographic Supplier*, that represents a repository of patient information that can be searched on demographic or visit-related fields.
The PIX profile involves the actor *Patient Identity Cross-reference Manager,* that serves a well-defined set of Patient Identification Domains. Based on information provided by in each Patient Identification Domain by a Patient Identification Source Actor, it manages the cross-referencing of patient identifiers across Patient Identification Domains.

The Application in the Heartfaid Platform
Following the transactions of the PIX and the PDQ integration profiles the middleware integrates the MPI module, for managing demographic information of the patients and guaranteeing their unique identification in the environment (domain) defined by the Heartfaid platform. Actually by integrating the MPI module in the middleware, the Heartfaid platform can be successively integrated with other Healthcare Information Systems safeguarding the centrality of the

patient and the standard mechanism for the interoperability with other legacy applications and solutions.

In the Heardfaid platform the MPI plays the role of the *Patient Identity Cross-reference Manager* actor, receiving the patient enrolment information from the web Portal, and the role of the *Patient Demographic Supplier* actor, replying to the Web Portal queries. The Web portal is, in fact, the main *Patient Demographic Consumer* actor, because it needs to associate information with a patient (in particular his MPI identifier) to interact with the other modules of the Heartfaid platform. It is important to underline that the Web Portal plays the role of the *Patient Identity Source*, because it is in charge to insert the information related to the patient enrollment.
The following picture shows a graphical schema of the interactions between the MPI module and the Web Portal in the Heartfaid platform, related to the implementation of the PIX and PDQ integration profiles.


HEARTFAID WEB PORTAL
Patient Identity Source
Patient Identifier Cross-Reference Consumer
Patient Demographic Consumer
Middleware
Patient Identity Cross-Reference Manager
Patient demographic Supplier
MPI

5.2.2 Cross-Enterprise Clinical Documents Share (XDS)

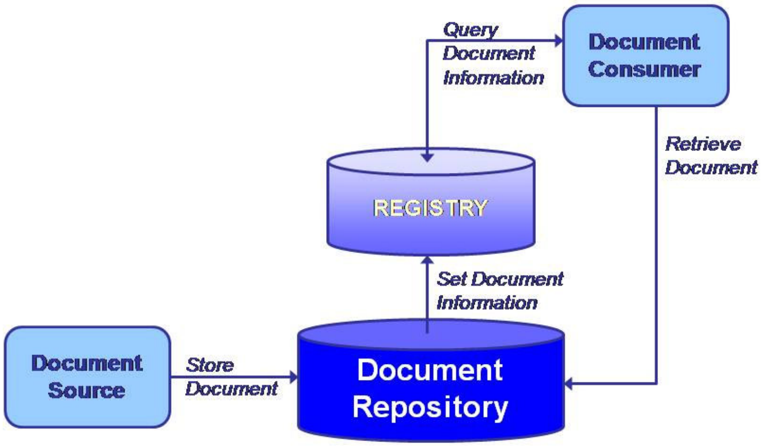
Description
The XDS profile facilitates the registration, distribution and access across health enterprises of patient electronic health records. XDS is focused on providing a standards-based specification for managing the sharing of documents between any healthcare enterprise, ranging from a private physician office to a clinic to an acute care in-patient facility.

Actors
The XDS profile involves the following actors:

- The *Document Source*: it is the producer and publisher of documents. It is responsible for sending documents to a *Document Repository* Actor. It also supplies *metadata* to the Document Repository Actor for subsequent registration of the documents with the Document Registry Actor.

- The *Document Repository*: it is responsible for both the persistent storage of these documents as well as for their registration with the appropriate *Document Registry*. It assigns a URI (Universal Resource Identifier) to documents for subsequent retrieval by a Document Consumer.
- The *Document Registry*: it maintains metadata about each registered document in a document entry. This includes a link to the Document in the Repository where it is stored. The Document Registry responds to queries from Document Consumer actors about documents meeting specific criteria. It also enforces some healthcare specific technical policies at the time of document registration.
- The *Document Consumer*: it queries a *Document Registry* Actor for documents meeting certain criteria, and retrieves selected documents from one or more Document Repository actors.

The following picture shows a graphical schema of the XDS integration profile.



The Application in the Heartfaid Platform

The Heartfaid middleware integrates in the platform a *Repository* module and a *Registry* module. These two modules allow to manage the information produced for the enrolled patients in the Heartfaid platform.

As described in the previous section, the implementation strategy followed for the implementation of the Heartfaid middleware has been to reduce (clear) the replication of the information produced by the different legacy (or developed in the Heartfaid project) applications integrated in the platform, by favouring the interoperability of the different modules. Anyway the Heartfaid platform generates new documentation for the enrolled patients, such as the responses of the Decision Support tools, the reports of advanced data and signal analysis, the combination of home monitored and clinical data (*real time reporting*) and so on. All this documentation will be inserted in the Repository of the Heartfaid platform, following the XDS integration profile.

Different modules of the platform can play the role of the *Document Source* actor, but in general the middleware integrates a special module (as described in the previous section) that is charge to collect information concerning clinical data,

DSS responses, alert messages and so on, compose a *xml* based message, store it in the Repository and insert the related metadata in the Registry module. This module allows to perform *real time reporting* and *forwarding reporting*.
Another key module of the Heartfaid Platform, a *Display*, will be in charge of retrieve the information stored in the repository for the presentation. In general, the Display module will be able to access also the data stored in all the database of the integrated modules.
It is important to underline that the using the profile PIX, PDQ, XDS and well defining the role of the Display, we have the key elements for the implementation of a EHR related to the Heartfaid Domain. Making these profile conform to the IHE integration profile, we can use in the future the Heartfaid platform as a new entity of distributed, integrated and heterogeneous Healthcare Information Systems.

5.2.3 Cross-Enterprise Sharing of Medical Summaries (XDS-MS)

Description

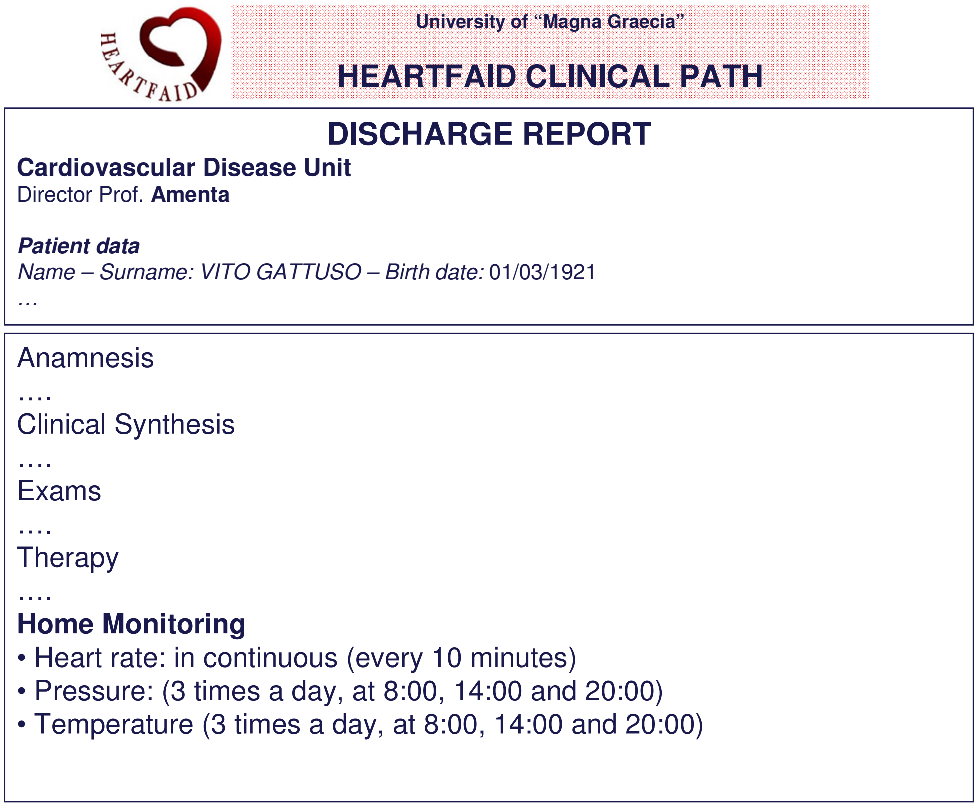
The XDS-MS profile represents a mechanism to automate the sharing process between care providers of Medical Summaries, a class of clinical documents that contain the most relevant portions of information about the patient intended for a specific provider or a broad range of potential providers in different settings. Medical Summaries are commonly created and consumed by electronic medical record systems at points in time of transfers of care such as referrals or discharge.

The Actors

The *Content Creator* actor creates content in an operative environment; there are different scenarios where the profile can be applied, such as the Discharge Summary operation, the Referral procedures, a general *Functional Status Assessment* and so on. The *Content Consumer* actor consumes content in the related environment.

The Application in the Heartfaid Platform

The objective of the Heartfaid project is to promote the Heartfaid platform as a crucial Information System for the integration between the homecare and the hospitals (secondary care), becoming itself a key point for care providing. For example, at the moment of the discharge of the patient, for the enrolled patients the Heartfaid platform could generate a more detailed discharge report, containing the suggested equipment for the home monitoring, a summary of the responses of advanced signal and imaging analysis and so on.
An example is shown in the following picture:



University of "Magna Graecia"

HEARTFAID CLINICAL PATH

DISCHARGE REPORT

Cardiovascular Disease Unit
Director Prof. **Amenta**

Patient data
Name – Surname: VITO GATTUSO – Birth date: 01/03/1921
…

Anamnesis
….
Clinical Synthesis
….
Exams
….
Therapy
….

Home Monitoring

- Heart rate: in continuous (every 10 minutes)
- Pressure: (3 times a day, at 8:00, 14:00 and 20:00)
- Temperature (3 times a day, at 8:00, 14:00 and 20:00)

In general, the Heartfaid platform could define new Functional Status Assessment, form managing the transfer of assessment information between practitioners during transfers of care across enterprises or from the healthcare structures to the home environment. In this case, the module of the Heartfaid platform that is in charge of managing the real time and the forwarding reporting will play the role of Content Creator actor, whereas the Content Consumer will be played from other system, such as the Electronic Medical Record of the General Practitioners.

5.2.4 Audit Trail and Node Authentication (ATNA)

Description

The ATNA profile describes the security environment (user identification, authentication, authorization, access control, etc.) assumed for the node so that security reviewers may decide whether this matches their environments, defines the basic security requirements for the communications of the node using TLS or equivalent functionality and defines the basic auditing requirements for the node.

Actors

The profile involves three actors: the *Audit Record Repository*, that is a system unit that receives and collects audit records from multiple systems, the *Secure Node*, a system unit that validates the identity of any user and of any other node and determines whether or not access to the system for this user and information exchange with the other node is allowed; it maintains moreover the correct time; the *Secure Application*, that guarantee the security aspects at application level.

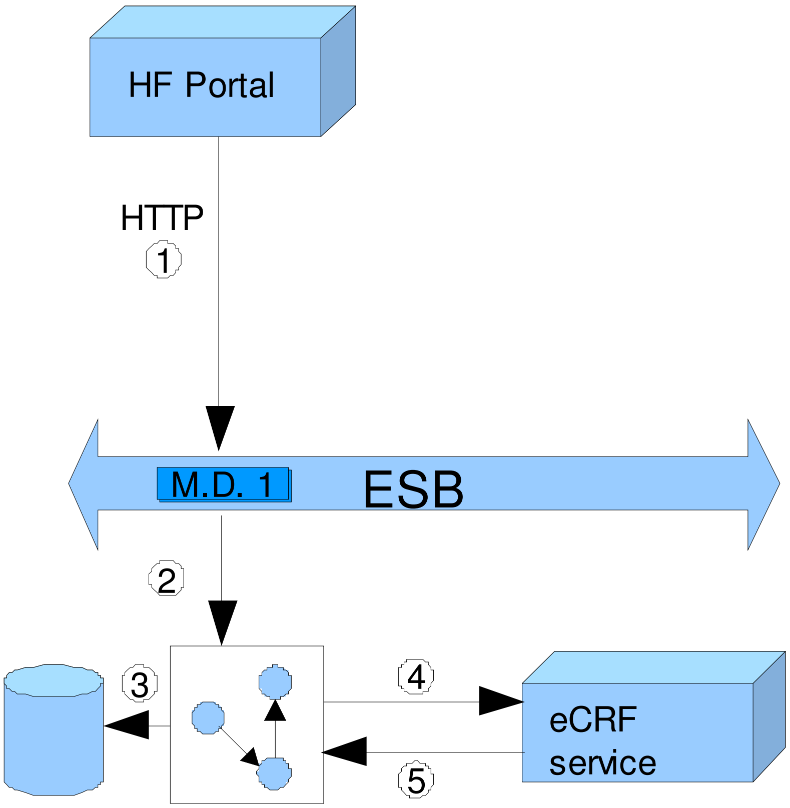
The Application in the Heartfaid Platform

With the ATNA profile the Heartfaid platform introduces security aspects in the management of the data of the patients and of the related transactions.

6. Reference scenarios

This section will describe, by using graphical representations, the most important workflows that have been implemented as composition of the functionalities described in Section 4.

6.1 New patient enrolment

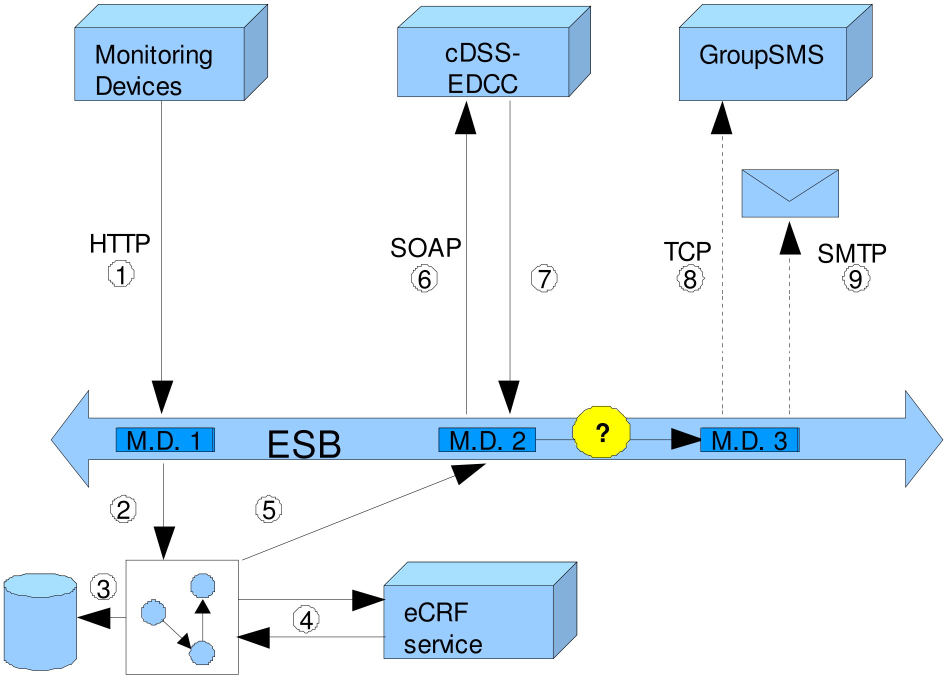


1	The message that is sent to the ESB	<?xml version="1.0" encoding="UTF-8"?> <request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd"> <action>patient.enrol</action> <payload> <record> <parameter name="patient.given">Vito</parameter> <parameter name="patient.family">Gattuso</parameter> <parameter name="patient.mother">Lorusso</parameter> <parameter name="patient.birthday">12</parameter> <parameter name="patient.birthmonth">1</parameter> <parameter name="patient.birthyear">1970</parameter> <parameter name="patient.gender">M</parameter> </record> </payload> </request>
M.D. 1	Mule descriptor for routing the information to a MIDA Graph that handles it.	<model name="HF-Model"> <mule-descriptor name="HFPortalMessages" implementation="com.synapsis.mida.mule.http.HTTPFrontendUMO"> <inbound-router> <endpoint address="http://localhost:3456" synchronous="true"> <filter className="org.mule.routing.filters.logic.NotFilter"> <filter pattern="/heartfaid/enrol,/heartfaid/searchpatient, /heartfaid/amiservices,/heartfaid/amienrol” className="org.mule.providers.http.filters.HttpRequestWildcardFilte"/> </filter> </endpoint> </inbound-router>

		<outbound-router> <router className="org.mule.routing.outbound.FilteringOutboundRouter"> <endpoint address="vm://mailer"/> <filter expression="syn:response/syn:ack/@refaction" className="org.mule.routing.filters.xml.JXPathFilter"> <properties> <property name="value" value="ami.enrol"/> <map name="namespaces"> <property name="syn" value="urn:SYN:SynThings"/> </map> </properties> </filter> </router> </outbound-router> <properties> <property name="urn-config-default" value="urnresolver.properties" /> <property name="urn-config-extra" value="com.synapsis.mida.flow.process.ProcessFactory.properties" /> <property name="mapping-resource" value="urn:file:mule:mapping.xml" /> <property name="xml-db-host" value="localhost" /> <property name="xml-db-port" value="8088" /> <property name="xml-db-user" value="heartfaid" /> <property name="xml-db-pass" value="heartfaid" /> <property name="xml-db-collection" value="/heartfaid" /> <property name="xml-db-context" value="/xmlrpc/db" /> </properties> </mule-descriptor>
2	A MIDA graph which is responsible for transforming the message, storing it to the MPI, and sending it to eCRF service.	... <Thing> <CompositeDefinition> <Input xlink:arcRole="input" xlink:href="InMarshaller:output"/> <Input xlink:arcRole="xsl" xlink:href="urn:file:conf:xslt:heartfaid:portal:requestadapter.xsl"/> <Input xlink:arcRole="message.uniqueid" xlink:href="MessageIDGenerator:output"/> <Input xlink:arcRole="patient.id" xlink:href="PatientIDGenerator:output"/> <Input xlink:arcRole="patient.iddomain">HF-Portal</Input> <Input xlink:arcRole="protocol.version">1.0</Input> <Input xlink:arcRole="sender.application" href="request:User-Agent"/> <Input xlink:arcRole="sender.facility">HF-Portal-Mule-Adapter</Input> <Input xlink:arcRole="receiver.application">syn:MPI</Input> <Input xlink:arcRole="receiver.facility">???</Input> <Identification> <Name value="Provider"/> </Identification> <Description> <Code reference="urn:syn:processor:xml:SYN:XSLTProcessor"/> </Description> <Output xlink:arcRole="output"/> </CompositeDefinition> </Thing> <Include xlink:href="urn:file:conf:flows:mpi:mpistore.xml"/> ... <Thing> <CompositeDefinition> <Input xlink:href="SavePatient:output" xlink:arcRole="synch"/> <Input xlink:href="DeleteNamespace:output" xlink:arcRole="input"/> <Input xlink:arcRole="method">post</Input> <Input xlink:arcRole="url">http://puls.cm-uj.krakow.pl:8080/heartfaid/interface.esb</Input> <Identification> <Name value="SendRequest"/> </Identification> <Description> <Code reference="urn:syn:processor:http:SYN:SimpleHttpInvoker"/> </Description> <Output xlink:arcRole="output"/> </CompositeDefinition> </Thing> ...

③	Message stored that gets stored into the MPI	<syn:Thing xmlns:syn="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" xmlns:hl7="urn:hl7-org:v3" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="urn:SYN:SynThings"> <syn:CompositeDefinition xlink:role="meta"> <syn:Identification> <syn:DocumentName>778a294f-e0e8-4f2e-9a41-fa513b97a197.xml</syn:DocumentName> </syn:Identification> </syn:CompositeDefinition> <syn:CompositeDefinition xlink:role="identificationContainer"> <syn:CompositeDefinition xlink:role="identification"> <syn:Identification> <syn:ID xsi:type="hl7:st">14f37170-46d3-40d5-b521-3cbe39c81c4d</syn:ID> </syn:Identification> <syn:Classification> <syn:Domain xsi:type="hl7:st">HF-Portal</syn:Domain> <syn:Encoding xsi:type="syn:isost">ISO</syn:Encoding> </syn:Classification> </syn:CompositeDefinition> </syn:CompositeDefinition> <syn:CompositeDefinition xlink:role="demographicContainer"> <syn:CompositeDefinition xlink:role="demographicData"> <syn:Identification> <syn:Given>Vito</syn:Given> <syn:Family>Gattuso</syn:Family> </syn:Identification> <syn:Classification> <syn:GenderCE code="M"/> <syn:Mother>Lorusso</syn:Mother> </syn:Classification> <syn:Quantification> <syn:BirthYear value="1970"/> <syn:BirthMonth value="1"/> <syn:BirthDay value="12"/> </syn:Quantification> </syn:CompositeDefinition> </syn:CompositeDefinition> </syn:Thing>
④	The message that is sent to the eCRF	<?xml version="1.0" encoding="UTF-8"?> <request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd"> <action>patient.enrol</action> <payload> <record> <parameter name="patient.id">14f37170-46d3-40d5-b521-3cbe39c81c4d</parameter> <parameter name="patient.given">Vito</parameter> <parameter name="patient.family">Gattuso</parameter> <parameter name="patient.mother">Lorusso</parameter> <parameter name="patient.birthday">12</parameter> <parameter name="patient.birthmonth">1</parameter> <parameter name="patient.birthyear">1970</parameter> <parameter name="patient.gender">M</parameter> </record> </payload> </request>
⑤	The answer of the eCRF	<response version="1.0"> <ack refaction="patient.enrol">AA</ack> </response>

6.2 Monitoring and Decision Supporting



①	A set of observations that flow from the monitoring devices.	<?xml version="1.0" encoding="UTF-8"?>
		    <gwph-message
		        xmlns ="http://www.synapsis.com/heartfaid/ph/dom"
		        xmlns:xsi ="http://www.w3.org/2001/XMLSchema-instance"
		                >
		        <destination-address>0</destination-address>
		            <sequence-counter>12</sequence-counter>
		            <compositeobs>
		             <param name="Telephone" type="string">(+30) 2810 391203</param>
		             <param name="ID number" type="string">33-333-33</param>
		             <param name="Surname" type="string">Zacharioudakis</param>
		             <param name="Sex" type="string">female</param>
		             <param name="Date of Birth" type="string">08/08/1980</param>
		             <param name="Occupation" type="string">Programmer</param>
		             <param name="First-name" type="string">Giorgos</param>
		             <param name="Social Security Number" type="string">22-222-22</param>
		             <param name="EHR-ID" type="string">1234</param>
		             <address descr="PC-Gateway" id="127.0.0.1"/>
		             <timestamp>2008-03-16T00:53:35</timestamp>
		             <obs>
		               <address descr="PC-Gateway" id="127.0.0.1">
		                <address descr="A&D BLOOD PRESSURE" id="0:a0:96:e:15:2b"/>
		               </address>
		               <param name="measurement-type" type="string">Supine</param>
		               <timestamp>{datetime1}</timestamp>
		                   </measurements>
		               <measurement>
		                           <type>Systolic BP</type>
		                           <value>121</value>
		                           <units>mmHg</units>
		                     </measurement>
		                     <measurement>
		                           <type>MAP</type>
		                           <value>119</value>
		                           <units>mmHg</units>
		                      </measurement>
		                      <measurement>
		                           <type>Diastolic BP</type>
		                           <value>127</value>
		                           <units>mmHg</units>
		                      </measurement>
		                      <measurement>
		                           <type>HR</type>
		                           <value>70</value>
		                           <units>bpm</units>
		                      </measurement>
		              </measurements>
		            </obs>
		           </compositeobs>
		</gwph-message>

M.D. 1	Mule descriptor for routing the information to a	<!-- Mule-descriptor with mida implementation for save sensor data -->
		  <mule-descriptor name="SensorMessages"
		          implementation ="com.synapsis.mida.mule.http.HTTPFrontendUMO">
		                      <inbound-router>
		                        <endpoint address="http://localhost:3458/heartfaid/sensor" synchronous="false"/>

	MIDA Graph that handles it.	</inbound-router>
		<!-- Return edcc message -->
		<outbound-router>
		  <router className="org.mule.routing.outbound.FilteringOutboundRouter">
		    <endpoint address="vm://edccChannel" />
		  </router>
		</outbound-router>
		<properties>
		  <property name="urn-config-default" value="urnresolver.properties" />
		  <property name="urn-config-extra"
		value="com.synapsis.mida.flow.process.ProcessFactory.properties" />
		  <property name="mapping-resource" value="urn:file:mapping.xml" />
		<property name="xml-db-host" value="192.158.100.156" />
		    <property name="xml-db-port" value="8088" />
		    <property name="xml-db-user" value="heartfaid" />
		    <property name="xml-db-pass" value="heartfaid" />
		    <property name="xml-db-collection" value="/heartfaid" />
		    <property name="xml-db-context" value="/xmlrpc/db" />
		</properties>
		</mule-descriptor>

②	A MIDA graph which is responsible for transofrming the observation to OGC, storing it to the eXist database, getting relevant clinical information related to the monitored patient and make a request to the cDSS EDDC service.	<ThingCollection xmlns:hl7="urn:hl7-org:v3" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="urn:SYN:SynThings" targetNamespace="urn:SYN:SynThings">
		...
		  <Thing>
		    <CompositeDefinition>
		    <Input xlink:href="SaveObservation:output" xlink:arcRole="synch"/>
		      <Input xlink:arcRole="input"
		        xlink:href ="urn:file:conf:xquery:sensor:createEDCCMessage.xqy"/>
		      <Input xlink:arcRole="idPatient"
		        xlink:href ="XQueryOnPatientsObservations:output"/>
		      <Input xlink:arcRole="xmlData" xlink:href="Provider:output" />
		      <Input xlink:arcRole="collection">heartfaid/data/observations</Input>
		      <Input xlink:arcRole="leftTimeDuration">P2DT4H30M</Input>
		      <Input xlink:arcRole="rightTimeDuration">P2DT4H</Input>
		      <Input xlink:arcRole="howMany">1</Input>
		      <Identification>
		        <Name value="CreateEDCCMessage" />
		      </Identification>
		      <Description>
		        <Code reference="urn:syn:processor:query:SYN:XQueryExecutor" />
		      </Description>
		      <Output xlink:arcRole="output" xlink:href="response:out"/>
		   </CompositeDefinition>
		  </Thing>
		...

③	The observation that gets stored into the eXist DB	<om:ObservationCollection xmlns:om="http://www.opengis.net/om">
		  <om:RichObservation>
		      <gml:name xmlns:gml="http://www.opengis.net/gml">PC-Gateway-127.0.0.1.A&D BLOOD PRESSURE-0:a0:96:e:15:2b</gml:name>
		      <gml:timeStamp xmlns:gml="http://www.opengis.net/gml">
		        <gml:TimeInstant>
		          <gml:timePosition>2008-03-16T00:53:35</gml:timePosition>
		        </gml:TimeInstant>
		      </gml:timeStamp>
		      <om:using xmlns:xlin="http://www.w3.org/1999/xlink" xlin:href="[127.0.0.1][0:a0:96:e:15:2b]"/>
		      <gml:target xmlns:gml="http://www.opengis.net/gml" xmlns:xlin="http://www.w3.org/1999/xlink" xlin:href="485cc627-80e6-433a-ae09-986bc63c94de"/>
		      <om:observable xmlns:xlin="http://www.w3.org/1999/xlink" xlin:href="Systolic BP"/>
		      <gml:resultOf xmlns:gml="http://www.opengis.net/gml">
		        <om:TypedQuantityList uom="mmHg">121</om:TypedQuantityList>
		      </gml:resultOf>
		    </om:RichObservation>
		...
		</om:ObservationCollection>

④	Query clinical data from eCRF	

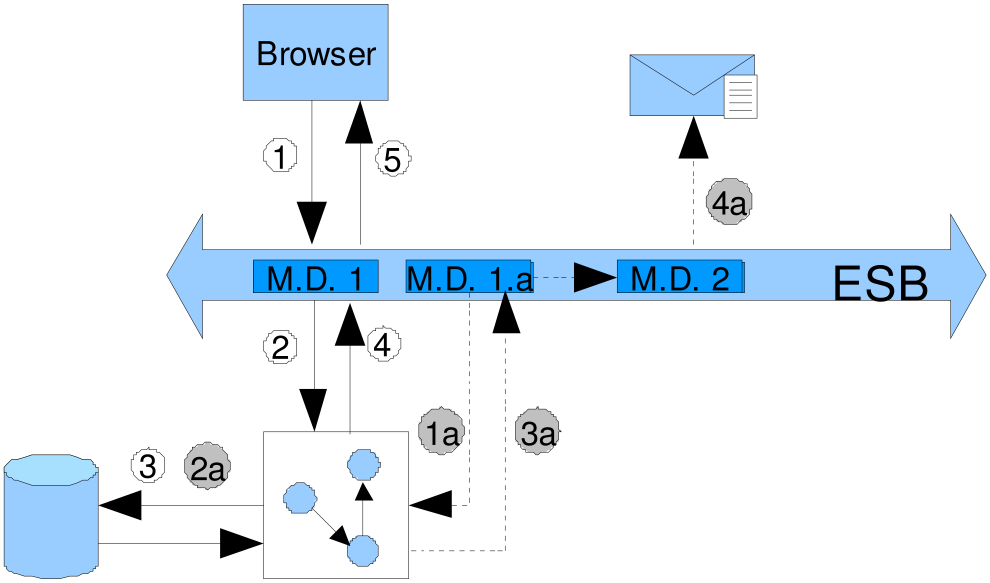
M.D. 2	The Mule descriptor that receives the message to be sent to the eCRF and routes it to the proper endpoint	<mule-descriptor name="CDSSTest" implementation="com.synapsis.heartfaid.cdss.CdssUMOInvoker">
		    <inbound-router>
		    <endpoint address="vm://edccChannel" />
		    </inbound-router>
		    <outbound-router>
		      <router className="org.mule.routing.outbound.FilteringOutboundRouter" >
		        <endpoint address="vm://PostCDSSDispatcher"/>
		      </router>
		  </outbound-router>
		</mule-descriptor>

⑤ ⑥	The message that is sent to the cDSS	<?xml version="1.0" encoding="UTF-8"?>
		<EarlyDetectionOfDecompensationConditions>
		  <Patient MPI="A001">
		    <Sex>male</Sex>
		    <Age>27</Age>
		    <NYHA>3</NYHA>
		    <Smoke>no</Smoke>
		    <Alcohol>0</Alcohol>
		    <CurrentSystolicBloodPressure>140</CurrentSystolicBloodPressure>
		    <CurrentHeartRate>60</CurrentHeartRate>
		    <CurrentWeight>88</CurrentWeight>
		    <CurrentTotalBodyWater>17.0</CurrentTotalBodyWater>
		    <PreviousSystolicBloodPressure>135</PreviousSystolicBloodPressure>
		    <PreviousHeartRate>65</PreviousHeartRate>
		    <PreviousWeight>87</PreviousWeight>
		    <PreviousTotalBodyWater>17.5</PreviousTotalBodyWater>
		  </Patient>
		</EarlyDetectionOfDecompensationConditions>

⑦	The answer of the cDSS	<?xml version="1.0" encoding="UTF-8"?> <EDDC_advice> <Patient MPI="A001"> <Alert Level="2">call you doctor NOW!</Alert> <Suggestion accuracy="0.9051">very high risk of decompensation</Suggestion> </Patient> </EDDC_advice>

? M.D. 3	Mule performs the analysis and the transformation of the message got as answer from the cDSS and if it matches the required conditions sends an email message or an email message plus an sms request.	<!-- Mule-descriptor for alarm flow --> <mule-descriptor name="CDSSMessages" implementation="org.mule.components.simple.BridgeComponent"> <inbound-router> <!-- <endpoint address="http://localhost:3457/heartfaid/testcdss"/> --> <endpoint address="vm://PostCDSSDispatcher"/> </inbound-router> <outbound-router matchAll="true"> <router className="org.mule.routing.outbound.FilteringOutboundRouter"> <endpoint address="vm://cdssmailer"/> <filter expression="EDDC_advice/Patient/Alert/@Level" className="org.mule.routing.filters.xml.JXPathFilter"> <properties> <property name="value" value="1"/> </properties> </filter> </router> <router className="org.mule.routing.outbound.FilteringOutboundRouter"> <endpoint address="vm://cdsssmser"/> <filter expression="EDDC_advice/Patient/Alert/@Level" className="org.mule.routing.filters.xml.JXPathFilter"> <properties> <property name="value" value="2"/> </properties> </filter> </router> </outbound-router> </mule-descriptor> <mule-descriptor name="CDSSMailer" implementation="org.mule.components.simple.BridgeComponent"> <inbound-router> <endpoint address="vm://cdssmailer"/> </inbound-router> <outbound-router> <router className="org.mule.routing.outbound.FilteringOutboundRouter"> <endpoint address="smtp://mail.websynapsis.com?address=f.conversano@websynapsis.com" synchronous="false"> <properties> <property name="subject" value="Alarm"/> <property name="fromAddress" value="hf &lt;hfmiddleware@websynapsis.com&gt;"/> </properties> </endpoint> </router> </outbound-router> </mule-descriptor> <mule-descriptor name="CDSSSmser" implementation="org.mule.components.simple.BridgeComponent"> <inbound-router> <endpoint address="vm://cdsssmser" transformers="StringToMessageTransformer"/> </inbound-router> <outbound-router> <router className="org.mule.routing.outbound.FilteringOutboundRouter" > <endpoint address="tcp://gsms.forthnet.gr:9999" transformers="StringToByteArray"/> </router> </outbound-router> </mule-descriptor>
⑧	The request message send to GroupSMS for requesting an SMS service	<?xml version="1.0" standalone="no"?> <!DOCTYPE XML SYSTEM "http://gsms.forthnet.gr/sms.dtd"> <XML> <REQUEST>SENDMESSAGES</REQUEST> <CUSTOMER> <USERNAME>synapsis</USERNAME> <PASSWORD>*********</PASSWORD> </CUSTOMER> <SUBJECT>.......</SUBJECT> <RECIPIENTS> <PHONE>ZZZXXXYYY</PHONE> </RECIPIENTS> <MESSAGE> <TEXT>very high risk of decompensation</TEXT> </MESSAGE> </XML>
⑨	According to the level of alarm and to the alerting policies, an eMail could be send in place of or together with the SMS	

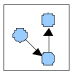
6.3 Reporting

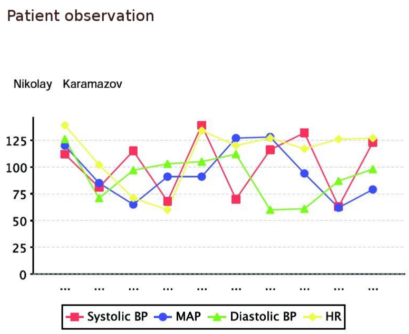


1	The message that is sent to the ESB for new patient report	<?xml version="1.0" encoding="UTF-8"?>
		<request xmlns="urn:SYN:SynThings" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:SYN:SynThings common.xsd">
		    <action>patient.report</action>
		    <payload>
		        <record>
		            <parameter name="patient.given">Vito</parameter>
		            <parameter name="patient.family">Gattuso</parameter>
		            <parameter name="patient.mother">Lorusso</parameter>
		            <parameter name="patient.birthday">12</parameter>
		            <parameter name="patient.birthmonth">1</parameter>
		            <parameter name="patient.birthyear">1970</parameter>
		            <parameter name="patient.gender">M</parameter>
		        </record>
		    </payload>
		</request>

M.D. 1	Mule descriptor for routing the information to a MIDA Graph that handles it.	<mule-descriptor name="MidaTest" implementation="com.synapsis.mida.mule.http.HTTPFrontendUMO">
		    <inbound-router>
		        <endpoint address="http://localhost:3444/TestReport" synchronous="true">
		            <properties>
		                <property name="Content-Type" value="application/pdf"/>
		            </properties>
		        </endpoint>
		    </inbound-router>
		    <properties>
		        <property name="urn-config-default" value="urnresolver.properties" />
		        <property name="urn-config-extra" value="com.synapsis.mida.flow.process.ProcessFactory.properties" />
		        <property name="mapping-resource" value="urn:file:mapping.xml" />
		  </properties>
		</mule-descriptor>

1a	The request of report might be generated by an internal module of the ESB, the same way as it was generated from the Portal by the end-user	
M.D. 1.a	Mule descriptor for a module responsible to request scheduled reports.	

2 1a

	A MIDA graph which is responsible for getting relevant clinical information related to the	<Thing>
		    <CompositeDefinition>
		        <Input xlink:arcRole="input" xlink:href="urn:file:conf:xquery:report:GetRichObservation.xqy"/>
		        <Input xlink:arcRole="idPatient">767a73f6-cf30-47bf-9dcf-8ffbc895fcf2</Input>
		        <Input xlink:arcRole="leftTimeDuration">P2DT4H30M</Input>
		        <Input xlink:arcRole="rightTimeDuration">P2DT4H</Input>
		        <Input xlink:arcRole="howMany">1</Input>
		        <Identification>

	monitored patient and creating pdf chart.	<Name value="GetRichObservation" /> </Identification> <Description> <Code reference="urn:syn:processor:query:SYN:XQueryExecutor" /> </Description> <Output xlink:arcRole="output" xlink:href="response:out"/> </CompositeDefinition> </Thing> ... <Thing> <CompositeDefinition> <Input xlink:arcRole="input" xlink:href="urn:file:conf:xquery:report:ObservationReport.jasper"/> <Identification> <Name value="InMarshallerJasper"/> </Identification> <Description> <Code reference="urn:syn:processor:report:utility:ByteArrayToJasperReport"/> </Description> </CompositeDefinition> <Output xlink:arcRole="output"/> </Thing> ... <Thing> <CompositeDefinition> <Input xlink:href="GetRichObservation:output" xlink:arcRole="input"/> <Input xlink:arcRole="jasper" xlink:href="InMarshallerJasper:output"/> <Input xlink:arcRole="outputmode">pdf</Input> <Identification> <Name value="ReportCreator"/> </Identification> <Description> <Code reference="urn:syn:processor:report:jasper:CreateReport"/> </Description> <Output xlink:arcRole="output" xlink:href="response:out"/> </CompositeDefinition> </Thing>
3 2a	Query to the Database	
3a 4a 4 5	The pdf chart	

7. Conclusions

The middleware prototype described in this document, succeeded his goal to guarantee the efficient access and exchange of all the patients' clinical data acquired both during home monitoring (raw data) and during his/her contacts with the health facilities (complex information).

In addition, the middleware developed guarantees that all the incoming, outgoing and exchanging information are compliant with the data representation and communication standards selected as suitable for the purposes of the project, as well as that all the communications performed among the internal modules of the platform and between the platform and the external applications are compliant with these standards. To this aim, common and well-known communication standards like Bluetooth, TCP/IP and HTTP have been adopted for implementing the interconnections between the medical devices and the Heartfaid systems and among the different software agents of the system itself. These have been designed in such a way to guarantee scalability and flexibility because of the large number of heterogeneous actors involved in the overall implementation of the platform.

It is obvious that, although the workpackage WP3 can be considered completed and the milestone "M3.3 – HEARTFAID Middleware prototype" successfully achieved, the prototype described in this document cannot be considered as final. On the contrary, it will be continuously tuned on the specific project needs that should arise during the testing and validation phases.

8. Bibliography and References

[1] **Data integration through database federation** by L. M. Haas, E. T. Lin and M. A. Roth

[2] **Enterprise Service Bus** by David A. Chappel. O'Reilly Theory in practice Series, June 2004 first edition

[3] IDC, *Integration Standard Trends in Program Development: It All Depends on What the Meaning of "Open" Is*, November 2003 (document #30365), *http://www.idc.com*

[4] **Enterprise Integration Patterns**, Designing, Building, And Deploying Messaging Solutions by G. Hohpe & B. Woolf. The Addison Wesley Signature Series

[5] http://mule.mulesource.org/display/MULE/Home

[6] http://www.python.org

[7] http://www.perl.org

[8] http://www.ruby-lang.org

[9] http://www.adobe.com/products/flash/

[10] http://java.sun.com

[11] http://java.sun.com/products/jms/

[12] http://www.w3.org/TR/**soap**/

[13] http://www.opengeospatial.org

[14]http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf

[15]http://www.oasis-open.org/committees/download.php/18617/wsrp-2.0-spec-pr-01.html

[16]http://www.jcp.org/en/jsr/detail?id=168

[17]http://www.jcp.org/en/jsr/detail?id=286

[18] http://www.jboss.org

[19] http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[20] http://www.w3.org

[21] http://celtix.objectweb.org/

[22] http://www.jasperforge.org/sf/projects/ireport

[23] http://www.iona.com/products/artix/

[24] http://www.sonicsoftware.com/index.ssp

[25] http://www.capeclear.com/