Remote Accessibility to Diabetes Management and Therapy in Operational Healthcare Networks

REACTION (FP7 248590)

# D5.6 REACTION SDK - Software Development Kit tools

Date 2013-02-28

Version 1.0

Dissemination Level: Public

# Table of Content

# Document control page

| | |
|---|---|
| **Code** | D5.6_REACTION_SDK_-_Software_Development_Kit_tools_v1.docx |
| **Version** | 1.0 |
| **Date** | 2013-02-28 |
| **Dissemination level** | PU |
| **Category** | P+R |
| **Document Owner** | CNET |
| **Participant Partner(s)** | CNET, ATOS, FIT, FORTH-ICS, FORTHNET, ALL |
| **Author(s)** | Stefan Asanin, Peter Rosengren, Peeter Kool, Tobias Brodén, Tamás Toth, Stelios Louloudakis, Matthias Enzmann, Carlos Cavero Barca, Ioannis Karatzanis, Ioannis Tsamardinos, Vincenzo Lagani, Franco Chiarugi. |
| **Work Package** | WP5 |
| **Fragment** | No |
| **Abstract** | This deliverable contains description of components available as tools in the REACTION SDK. |
| **Status** | ☐ Draft<br>☐ Ready for internal review<br>☒ Task leader accepted<br>☒ WP leader accepted<br>☒ Technical Manager accepted<br>☒ Project Coordinator accepted<br>☐ Other (please specify if checked) |
| **Previous Versions** | |

**Version Notes**

| Version | Author(s) | Date | Changes made |
|---|---|---|---|
| 0.1 | Stefan Asanin | 2013-01-18 | Initial ToC |
| 0.3 | Stefan Asanin | 2013-02-17 | Reformatted content |
| 0.7 | Stefan Asanin | 2013-02-28 | Checked document |
| 0.8 | Stefan Asanin | 2013-02-28 | Revised document |
| 0.9 | Stefan Asanin | 2013-02-28 | Revised document |
| 1.0 | Stefan Asanin | 2013-02-28 | Final version submitted to the European Commission |

**Internal review history**

| Reviewed by | Date | Comments made |
|---|---|---|
| Lasse Christiansen | 2013-02-28 | Approved with minor comments |
| Ivo Ramos | 2013-02-28 | |

# 1 Executive summary

The REACTION Software Development Kit (SDK) will allow developers to rapidly create new networked applications on the REACTION platform. The generalised platform will support cost-effective development of a broad range of innovative healthcare applications, so a user-friendly development platform is warranted. The SDK will provide solution developers with a high-level interface for innovative monitoring applications with embedded intelligence and closed loop feedback provisioning using the REACTION platform.

The REACTION SDK toolkit for model-driven development of applications that use the REACTION platform is a major tool for developing new telemedicine or eHealth applications or for device manufacturers wanting to make their devices interoperable.

# 2 Introduction

The REACTION project aims to develop an integrated approach to improve long term management of diabetes through continuous blood glucose monitoring, monitoring of significant events, monitoring and predicting risks and/or related disease indicators, decision on therapy and treatments, education on life style factors such as obesity and exercise and, ultimately, automated closed-loop delivery of insulin.

## 2.1 Background

The REACTION project seeks to use the great potential of new technologies to cope with the increasing number of citizens suffering from insulin-dependent diabetes. A user centred approach will be used focused on the involvement of all stakeholders (i.e. patients, relatives and professional carer as well as healthcare commissioners, business stakeholders, and regulatory authorities) in an iterative cycle with the intention of maximizing the probabilities of success of the new technological platform of services.

Technically, the REACTION platform (Figure 1) is structured as an interoperable peer-to-peer communication platform based on Service-Oriented Architecture (SOA) where all functionalities, including the measurement acquisition performed by sensors and/or devices are represented as services and applications. These consist of series of services that are properly orchestrated in order to perform a desired workflow. The REACTION platform also will make extensive use of dynamic ontologies and advanced data management capabilities offering algorithms for clinical assessment and evaluation.

Security and safety of the proposed services will be studied and necessary solutions to minimize risks and preserve privacy will be implemented. Legal framework for patient safety and liability as well as privacy and ethical concerns will be analysed and an outline of a policy framework will be defined. Moreover, impacts on health care organizations and structures will be analysed and health-economics and business models will be developed.

## 2.2 Purpose, context and scope of this deliverable

A range of REACTION services are being developed targeted to the management of insulin-dependent diabetic patients in different clinical environments. The services aim to improve continuous blood glucose monitoring (CGM) and insulin therapy by contextualized glycaemic control based on patient activity, nutrition, interfering drugs, stress level, etc. for a proper evaluation and adjustments of basal and bolus doses. Decision support will assist healthcare professionals, patients and informal carers to make correct choices about blood glucose control, nutrition, exercise and insulin dosage, and thus to reach a better management of diabetes therapy.

REACTION will further develop complementary services targeted at the long term management of all diabetic patients, Type I and Type II. Integrated monitoring, education, and risk evaluation will ensure all patients remain at healthy and safe blood glucose levels, with early detection of onset of complications.

The purpose of this deliverable is to provide and describe all the components that are used in the REACTION pilot but that also can be reused in other applications as part of the REACTION SDK. Each component comes as description of a tool that independently or jointly can be deployed and used to serve diabetes management and care.

# 3 Understanding the REACTION SDK hierarchy

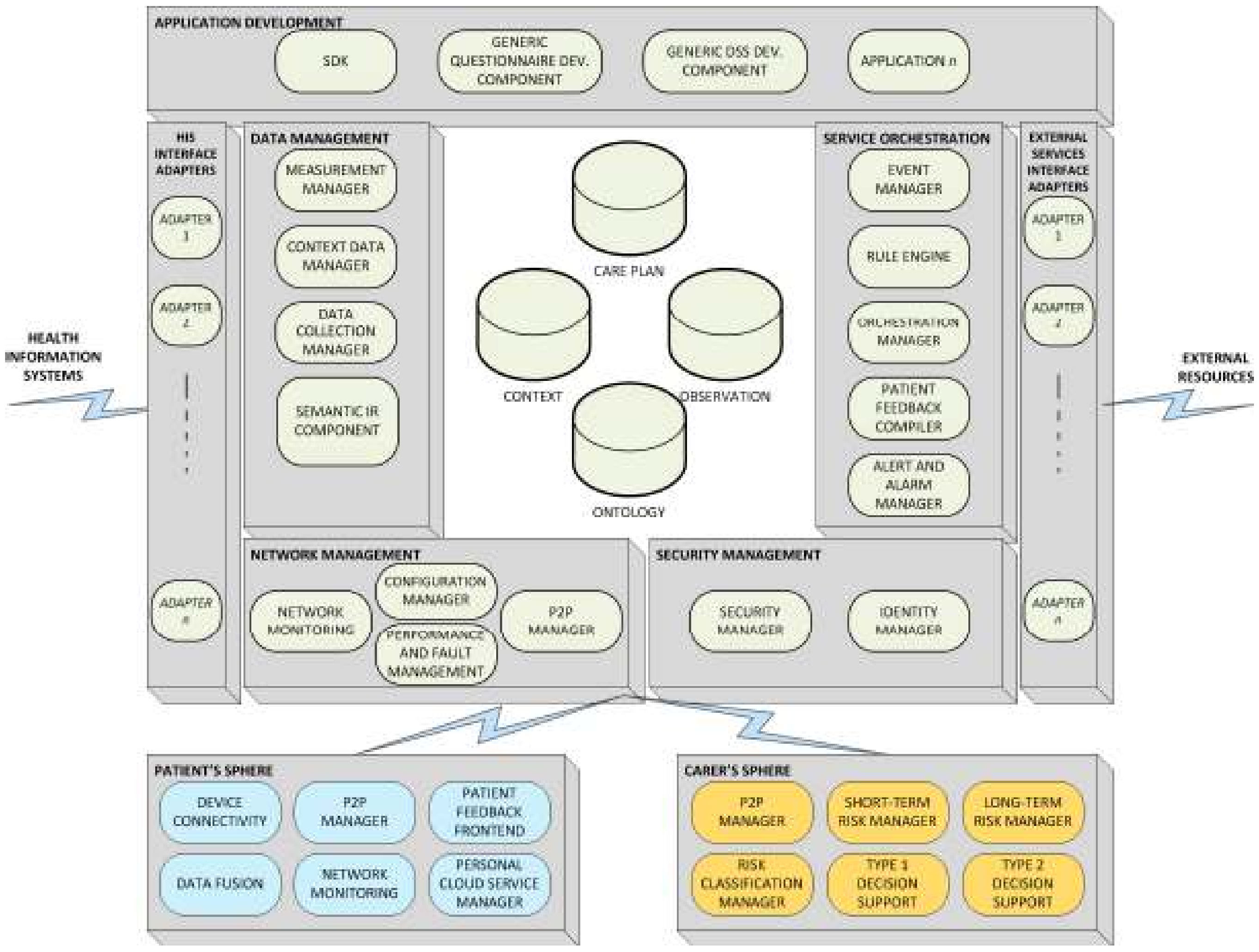


Figure 1: REACTION modules and enclosed components.

The REACTION platform is the central production environment for the deployment of REACTION applications. According to the DoW the platform consists in five subsets each one responsible for their part of the overall functionality. These subsets make up the encapsulating modules in Figure 1.

## 3.1 The REACTION platform subsets

1. The **Data Management** subset is central to the high level functioning of applications and services deployed on the platform. It implements the model-driven architecture for application development and deployment, the service oriented architecture for core service functionalities, data manipulation, data fusion and event handling. It also manages data transfer to and from nodes and stakeholders in a REACTION environment.
2. A **Service Orchestration** subset will orchestrate the different services available in a pre-described sequence for execution. This component introduces higher abstraction mechanisms and makes the application developer independent of using a specific programming environment to orchestrate REACTION applications.
3. The **Network Management** subset is responsible for the physical communication between devices, persons and external repositories. Each PAN node will have its own Network Manager and each Network Manager will have an external Web Service based interface where it can exchange data with remote Network Managers.

4. The **Security Management** subset will perform mapping and brokering between security models, user and client devices profiling management, mapping and usability between trust domains, and semantic standards and generalisation ontologies development.
5. The **Application Development** subset is an open SDK for model-driven development of applications that use the REACTION platform.

The purpose is to let applications to be developed and deployed to execute comprehensive tasks. Each application serves specific goals and is constructed from a series of standardised workflows and business rules. Applications are presumed to be developed and stored in the form of conceptual domain models (ontologies). The domain model describes the functionality, the objects involved (devices, users, rule sets, repositories, etc.), the security model to be used and the run-time environment. Thanks to the REACTION SDK, applications are easy to build, modify and deploy to different features.

## 3.2 How we make the components work together

The model-based application development and scalability approach taken in REACTION set an architecture that supports model-driven development of services. It will allow service providers to rapidly build, maintain and update services operating on the REACTION platform. The REACTION development platform has the aim to be an open toolkit used for model-driven development of services that use the REACTION platform and will be based on a structure of service ontologies. A conceptual domain model describes the application, the services to be deployed and the objects involved (devices, users, rule sets, repositories, etc.). A domain model is mapped to an operative data model, which is implemented by an XML schema and a set of Web Service interfaces. Each externally accessible component provides a WSDL interface, which exposes a subset of the domain model XML schemas. Device specific services can be integrated with external services, such as knowledge extraction, accessing an Electronic Patient Record (EPR) or providing feedback to a carer, merged with workflow and resource scheduling services and supplied with security model and authentication services. The deployment of applications will take the form of high-level and dynamic orchestration of individual services. The actual service will then take the form of instances of basic services requiring only personalisation and instantiation of objects and parameters.

The starting point of the iterative design process was a set of domain-specific vision scenarios delivering end-user visions of applications in three different insulin therapy domains: General Ward, Outpatient and Automatic Glucoses Control. This document will try to compile all these domains in order to provide a single unified technical representation that is able to describe the development of the REACTION platform where a SOA-based framework is used for domain-specific application developers.

The JIRA Volere requirements created and made available online for the REACTION consortium illustrates a wide area of applicability of the platform components. The intention was to make these requirements to lead the way for the main functions that are defined. Initially we also have predefined and already fully or partly available components that need to be assigned with proper roles and functionalities.

## 3.3 Online resources

In the course of the project, repositories were designed and made available to the involved partners in order to facilitate the exchange of software components and artefacts and the joint development of software.

Software components and artefacts can be exchanged using an FTP secure server repository accessible through FileZilla (ver. 3.5 or higher).

The repository is hosted by FORTH at the link: ftpes://thor.ics.forth.gr/ and configuration (connectivity, accounts, security) and maintenance (including user management) is performed by FORTH upon agreement with the REACTION consortium.

A screenshot of such repository is illustrated in fig. 1.

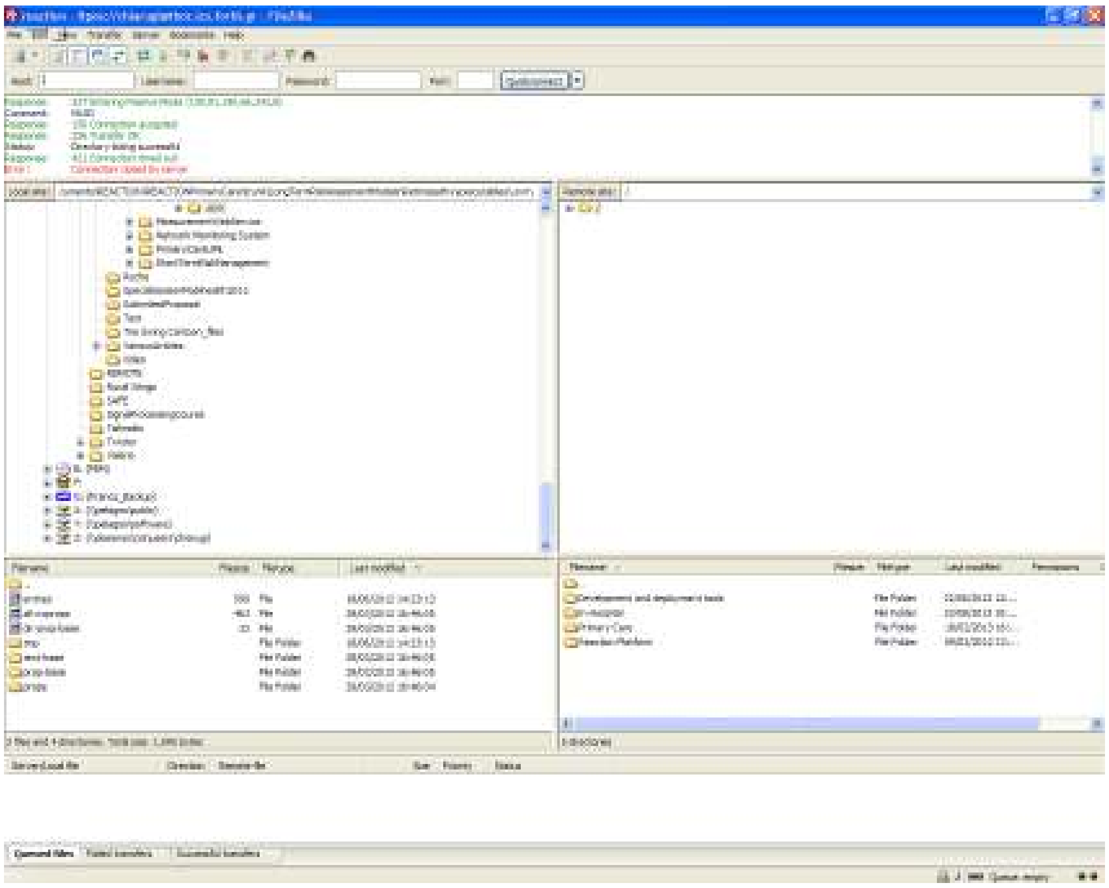

Development and deployment tools
In-Hospital
Primary Care
Reaction Platform

Figure 2 Access to the FTP repository through FileZilla.

For components developed by more than one partner a repository with versioning (SVN) has been made available.

The repository contains 3 different folders for the development of 3 different application environments (in-hospital, primary care and REACTION platform).

The repository is hosted by FORTH and can be accessed through Tortoise SVN using the links:

https://thor.ics.forth.gr/svn/Reaction (for the in-hospital)

https://thor.ics.forth.gr/svn/ReactionPrimaryCare (for primary care)
https://thor.ics.forth.gr/svn/ReactionPlatform (for the generic platform)

Configuration (connectivity, accounts, security) and maintenance (including user management) is performed by FORTH upon agreement with the REACTION consortium.

A couple of screenshots of these repositories (once accessed with TortoiseSVN) are shown in fig. 2 and 3.

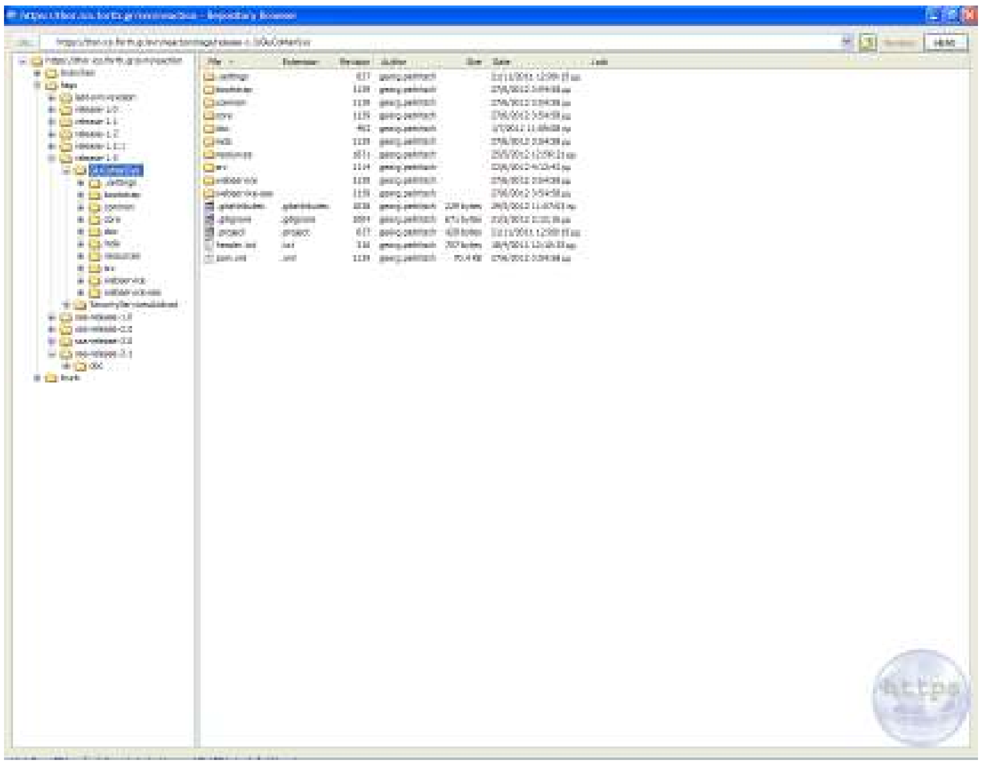


Figure 2: Access to the SVN repository (for in-hospital) through TortoiseSVN.

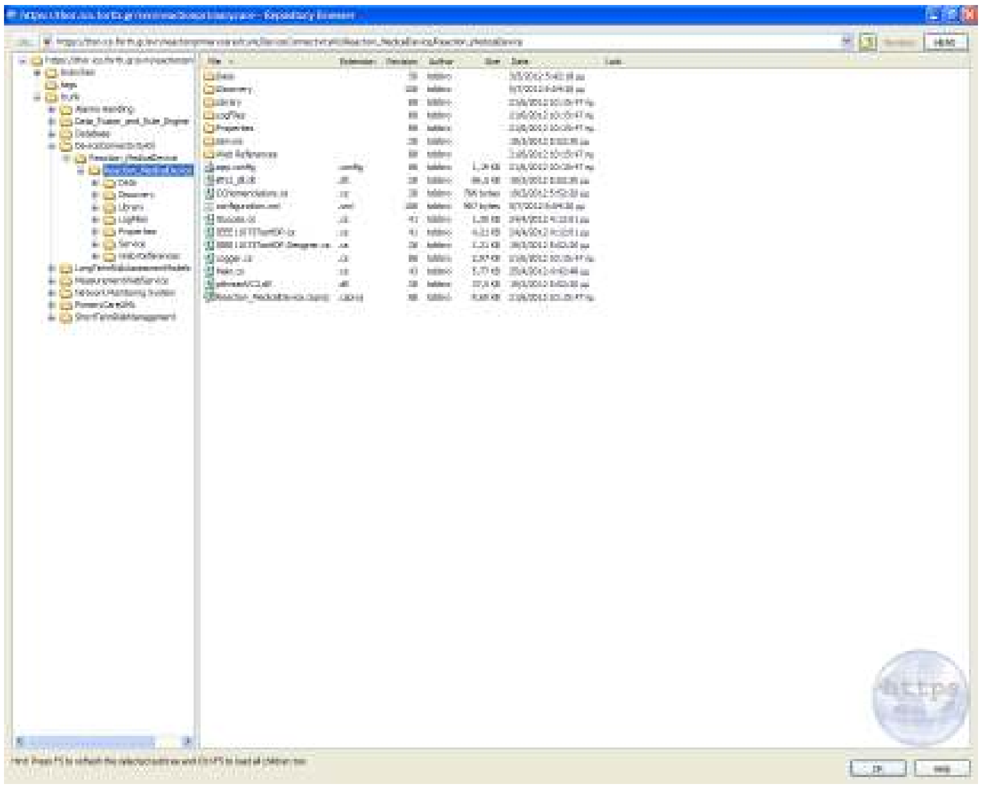


Figure 3: Access to the SVN repository (for primary care) through TortoiseSVN.

These repositories will remain available for the consortium partners for at least a year after the end of the project.

# 4 Data Management

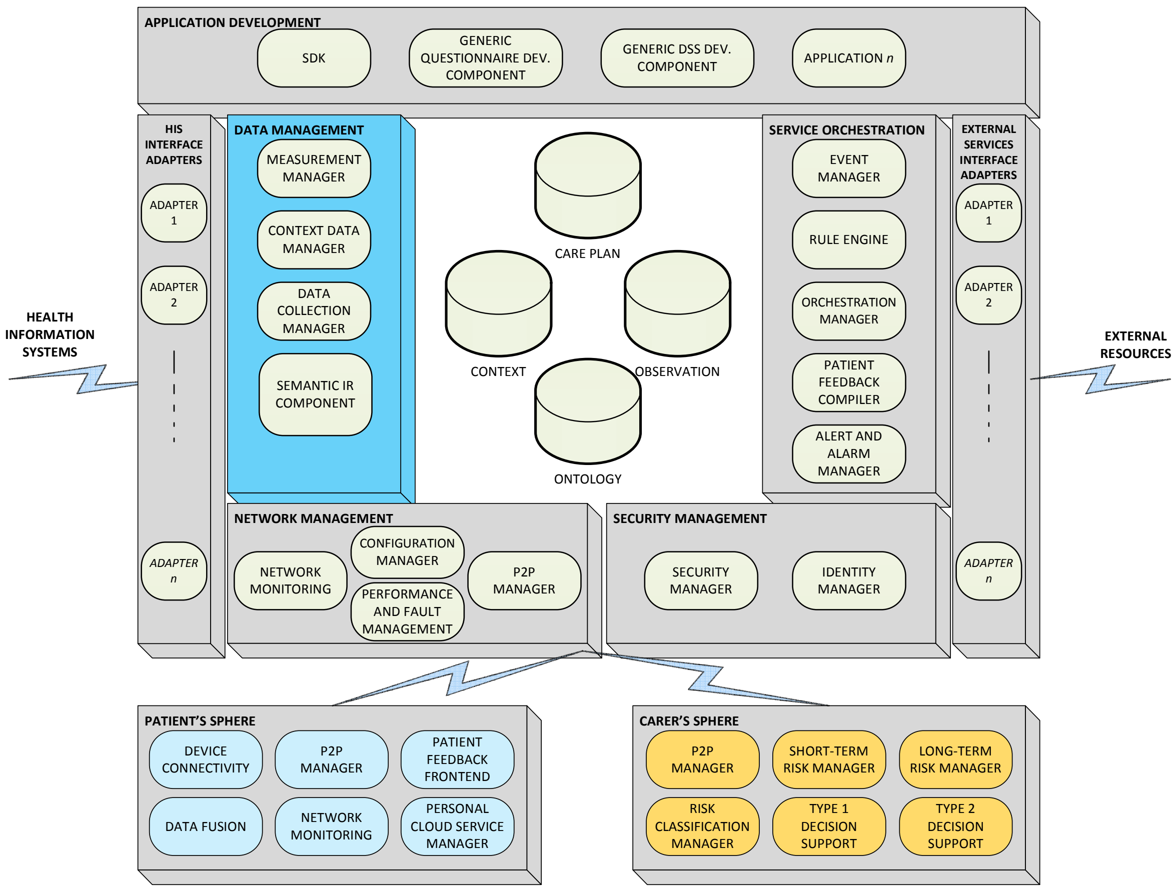


Figure 3: The Data Management subset of the REACTION platform.

The need for semantic interoperability is driving developments in current healthcare information technology, and it is an important goal of REACTION. Semantic interoperability is about the clear understanding of stored, used and communicated data and information by the users of this information, in particular patients and health care professionals.

It is defined as: "Semantic interoperability means ensuring that the precise meaning of exchanged information is understandable by any other system or application not initially developed for this purpose” (EC Recommendation, COM (2008) 3282 final). Semantic interoperability implies that received data can be combined seamlessly with local data and processed homogeneously, and that clinical data from local and external systems can be combined and processed similarly and collectively without loss of meaning. This requires the unambiguous definition of any used concepts where our basis for semantic interoperability in REACTION will be based on the use of different ontologies.

## 4.1 Measurement Manager

The Measurement Manager provides functions for updating, querying and retrieving measurements from the Observations Database.

Main functions are:

- Receive measurements and update database.
- Retrieve different vital signs measurements (e.g. blood Glucose, blood pressure, weight scale $SpO_2$, etc.) for patient Retrieve different vital signs measurements for specified time period.
- Provide conversions of measurements to different formats (e.g. ORU to XML and back).

- Delete measurements.
- Checks measurement so it complies with patient informed consent.
- Acknowledge the received and successful storage of data.

## 4.2 Context Data Manager

The Context Data Manager provides functions for receiving, updating, deleting and retrieving context data (nutrition, lifestyle, physical activities) from the context database.

Main functions:

- Retrieve context associated with a patient and a set of measurements.
- Receive and store context associated with a patient and environmental data.
- Update context and delete context associated with a patient.

## 4.3 Data Collection Manager

The Data Collection Manager is responsible for receiving incoming measurements and context data and relates this to the correct patient. It routes the incoming messages to the correct manager e.g. Measurement Manager or Context Data Manager. It acts as a frontend gate for all externally incoming messages.

Main functions are:

- Exchanges basic configuration information with client gateways, e.g. what devices should be there
- Adjusts transfer frequency after its performance quality
- Provides transparent communication to and from clients and other components
- Receive measurement
- Receive context data
- Receive device events and info
- Receive patient info
- Relate measurement to patient
- Relate context to patient
- Publish abnormal device state (events)
- Relate incoming patient info to stored patient
- Keep incoming message queue(s)
- Provide audit trail by logging incoming messages
- Unpacks data fused collections and updates context and measurement databases accordingly
- Provides client gateway statistics, e.g. number of messages processed per day.
- Checks with client gateways which devices are active and their QoS.

## 4.4 Semantic IR Component (SIR)

The Semantic Information Retrieval Component (SIR) enables the user to search for information in textual documents. Three different kinds of archives can be searched:

- data base of EPRs – since EPR usually contains data represented in the form of natural language text,
- the Cochrane archive,
- archives of guidelines

The first priority is the search in the patient records.

Figure 4 provides an overview on the components of the SIR.

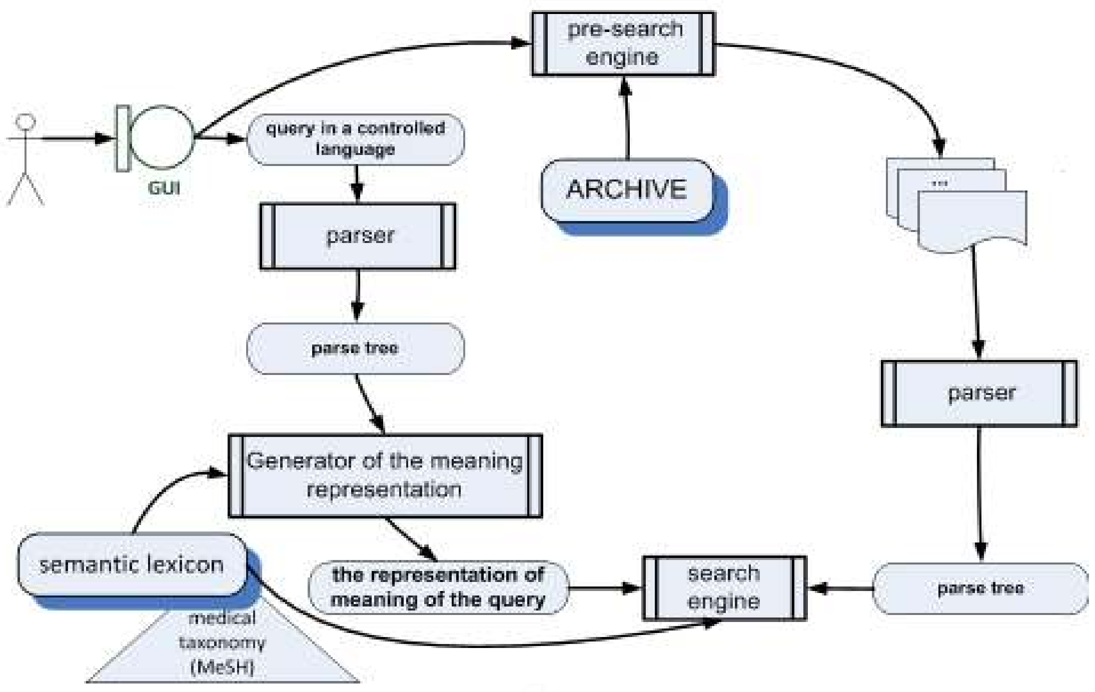


Figure 4 Structure of SIR

The following subsections describe the components.

## GUI

The GUI enables the user to enter the query, keywords and other data for the retrieval of relevant documents, and shows the result of the search. The GUI:

- helps the user to edit the query in the form of phrases and/or sentences of a controlled natural language
- helps the user to give keywords and other data for the search,
- verifies whether the query satisfies the rules of the controlled language, and gives warnings and error messages,
- shows the list of the documents resulted by the search in an ordering according to their relevance,
- after selecting a document shows the phrases/sentences having meaning similar to the query.

## Parser

The Parser creates a graph (DAG – directed acyclic graph) representation of the syntactic structure of the query and of segments of natural language texts. The Parser analyses natural language phrases/sentences.

The Parser consists of several modules as it is usual in natural language processing (NLP). Some of them are standard NLP programs; however, the most important components are made for the semantic search according to special requirements.

Parsing the query or segments of documents is somewhat different, because the inputs are different. In the first case a pre-processed and structured version is the input, in the second case the input is the text itself. Therefore while in the first case the correctness of parsing is ensured, in the second case it is not. However, problems in the parsing of text segments do not cause failures in the search, because of the strategy built into the search engine.

### Semantic Lexicon

The Semantic Lexicon stores the lexical units (words) with their semantic properties. The Semantic Lexicon provides data for the Generator of the Meaning Representation. The Semantic Lexicon consists of several components. There are separate but connected storage for the predicative words (words expressing events, relations, mainly verbs) and for non-predicative ones. The reason is that different resources store them. The predicative words are obtained from FrameNet; however, the FrameNet corpus has to be completed substantially. WordNet is used for the non-predicative words. Above these stores of words there are ontology capsules acting as upper ontology segments. The medical terminology is obtained from MeSH.

The words are grouped into synonym-sets (synsets). We use the notion of being synonym in a less strict way than it is used in linguistics: two words are synonym, if they refer to similar situation or item. This definition is adequate for information retrieval. With the predicative words their valence patterns are also represented, including semantic features (role relations) of the valence units. The role relations are connected to the relations defined in the ontology capsules.

### Generator of the Meaning Representation

The Generator of the Meaning Representation is responsible for building a meaning representation of the query from the result of the Parser. The Generator of the Meaning Representation builds the meaning representation of the query using the data obtained from the semantic Lexicon. The Generator of the Meaning Representation:

- traverses the DAG got from the parser and
- represents words by their synsets,
- connects the synset tokens by relations corresponding to the syntactical connections of the words

### Pre-search Engine

The Pre-search Engine substantially reduces the amount of documents to be processed by a preliminary search. The Pre-search Engine executes a key-word based search. If data base of EPRs are searched, it executes the data base search based on numerical and coded data.

### Search Engine

The Search Engine finds the phrases that may have similar meaning as the query (or its part) has. The Search Engine tests segments of the syntactical structure of texts, whether they corresponds to segments of the meaning representation of the query.

# 5 Service Orchestration

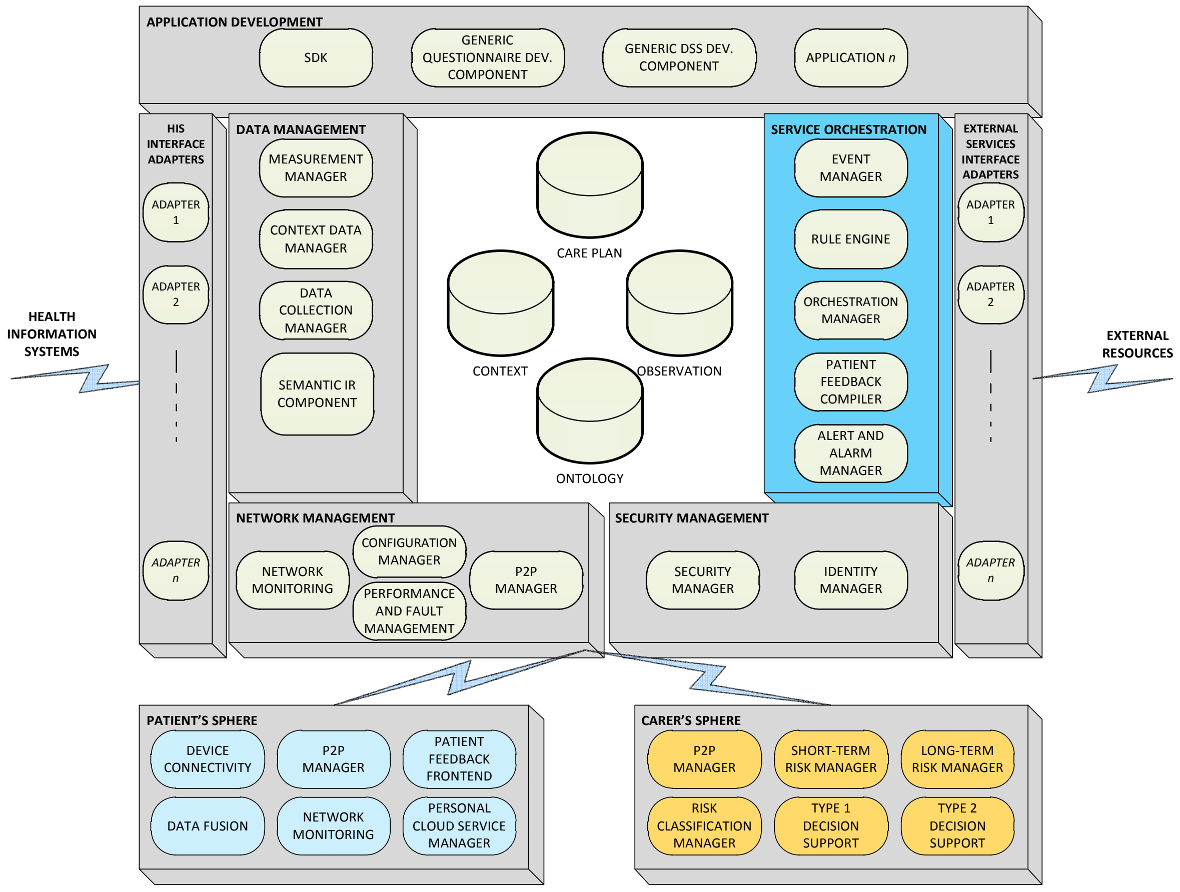


Figure 5: The Service Orchestration subset of the platform.

Ensembles of REACTION services are orchestrated by a specific high-level workflow e.g. based on Business Process Execution Language (BPEL). The workflow will be specified in the application and interpreted by the Orchestration Manager. The Orchestration Manager will make sure the different services available are executed in a pre-described sequence. This component introduces higher abstraction mechanisms and makes the application developer independent of using a specific programming environment to orchestrate REACTION applications. It will also eliminate the interdependencies of services, solve conflicts of services and provide the most flexibility environment needed to realise service-oriented applications. The Orchestration Manager will also interface with legacy back end systems such as operational workflow and resource scheduling systems (SAP, EPR systems).

## 5.1 Event Manager

The Event Manager is responsible for providing, publishing and subscribing functionality to the REACTION platform. In general, publish/subscribe communication provides an application-level selected multicast that decouples senders and receivers by time, space, and data (i.e., sender and receivers do not need to up at the same time, do not need to know each other's network addresses and do not need to use the same data schema for events they send). The Event Manager will be used in any place where there is a potential many-to-many relationship between senders and receivers and where asynchronous communication is desirable.

The Event Manager provides publish/subscribe functionality, i.e., the ability for publishers to send a notification to multiple subscribers while being decoupled from them (in terms of, e.g., not holding direct references to subscribers). The specific variant of publish/subscribe implemented is topic-based publish/subscribe where key/value pairs represent events. With this approach, any subscriber or publisher defines a topic simply by executing the “publish” or “subscribe” actions. In particular, the Event Manager provides the following main functionalities:

- Subscription support allowing clients to subscribe to published events via a topic-based publish/subscribe scheme
- Publication support allowing client to publish event on topics
- Routing events to subscribed clients
- Event Core manages persistent subscriptions, publication to subscription matching etc.
- Interfacing to Network Manager (e.g., broadcast-, multicast-, or gossiping-based dissemination)
- Storing events
- Priorities events
- Retry sending events.

The Event Manager interface provides the methods for handling all the subscriptions, notifications and publications. Figure 6Figure 6 provides an overview on the internal components of the Event Manager.

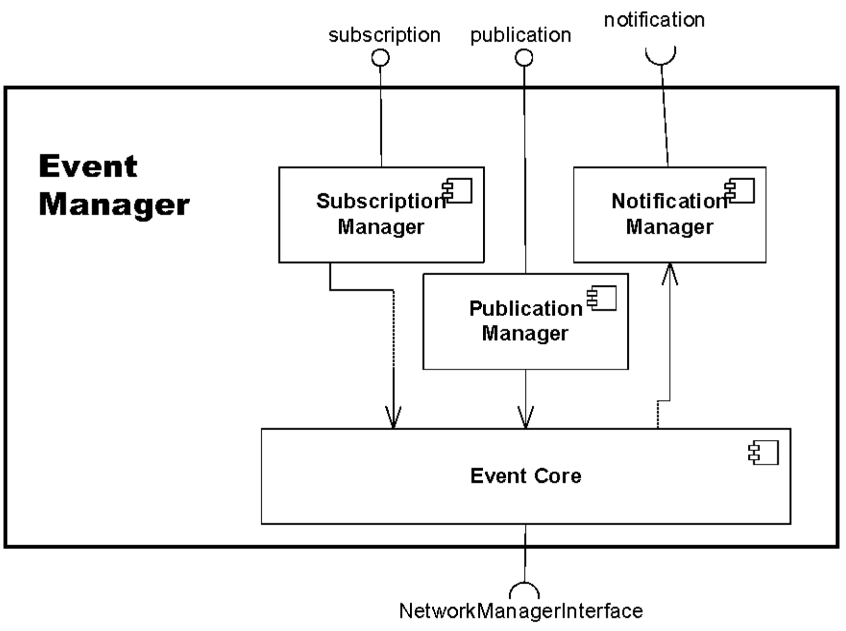


Figure 6: Event Manager.

The following subsections describe how the responsibilities are distributed between the internal managers of the Event Manager.

### Subscription Manager

The Subscription Manager is responsible for handling subscriptions, providing the methods: subscribe, subscribeWithHID, unsubscribe, unsubscribeWithHID, getSubscriptions, clearSubscriptions and clearSubscriptionsWithHID. The main functionalities of the Subscription Manager are the following:

- Handle (add, consult, remove) subscriptions.

This manager is responsible for handling subscriptions. This manager will provide the methods: *subscribe, unsubscribe, getSubscriptions, clearSubscriptions*.

**Notification Manager**

The Notification Manager is responsible for notifying the subscribers of a specific topic that an event with the same topic was published to the Event Manager. The main functionalities of the Notification Manager are the following:

- Notify all the subscribers of a specific topic, that and event was published.

The Notification Manager component is responsible for notification of subscribers of an event. This notification is handled by a queue of threads. This queue of threads is ordered by topic priorities, so if several publications arrive to the Event Manager, the notification of the subscribers is done in a prioritized way, depending on the priority of a topic, e.g., if topic A has higher priority than topic B, all the subscribers of topic A will be notified first of this event. The notification of a subscriber can be done directly to the subscriber in case the subscription was made using the “subscribe” method, or the notification can be done through the Network Manager, in case the subscriber invoked the method “subscribeWithHID”.

**Publication Manager**

The Publication Manager is responsible for handling publications to the Event Manager. The main functionalities of the Publication Manager are the following:

- Handle publications from publishers.

The Publication Manager component is responsible for receiving publications from the publishers, and delivers this publication to the Notification Manager that will notify all the subscribers to the event topic published with the event data.

**Event Core**

The Event Core component will contain the subscription, subscribers and topics data structures and it will provide the interface to the Network Manager. The Event Core component has the following main functionalities:

- Contain subscriptions data
- Contain subscribers’ data
- Contain topics data

The Event Core component is the component containing all the data structures needed to be consulted, changed by the other sub components of the Event Manager (Publication Manager, Subscription Manager, and Notification Manager).

## 5.2 Rule Engine

The Rule Engine is responsible for managing and executing a set of rules. A rule is triggered when a certain condition is met. The rule triggering leads to that a specified action is taken. This includes rules for monitoring vital signs measurements but the Rule Engine is not limited to this. The rules are flexible and allow specifying thresholds, targets and intervals for vital signs.

Therefore, the Rule Engine is consequently flexible and allows expressions of very sophisticated rules although it is not expected that clinicians will use this interface. But rather, they will use the graphical user interface described in the next coming sections.

Main functions are:

- Add rule
- Update rule
- Delete rule
- Evaluate rules
- Get rule report
- Get patient rules
- Associate alarms and alerts with patient rules

The Rule Engine in this second prototype is implemented as an IoT (Internet of Things) -enabled device using the Hydra middleware and it follows the same principles as the Data Fusion Engine. This means it is possible to have a number of “virtual devices” doing specific rule tasks.

Such a Rule Engine device is configured using two configuration files. The first file defines the events which define the scope of the rule device, i.e. these are the events that can trigger an action from this rule engine device. The format for the event file is simple:

.<events>

<event>newobservation</event>

</events>

The above example defines that this rule engine will listen to the event “newobservation”.

The second file defines the rules and the actions to take if a rule triggers. The rules are expressed using XSL-T. The following shows an example of a rule device that processes new incoming observations for blood pressure and weight. The action it takes is to send an SMS to the patient to confirm that the measurements have been received and processed.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:SMS="urn:SMS" xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:user="http://www.cnet.se" xmlns:cnetuser="http://www.visualnetserver.com"  exclude-
result-prefixes="SMS msxsl user">
  <xsl:output method="xml" version="1.0" encoding="ISO-8859-1"/>
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>
<xsl:template match ="event[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']">
<xsl:variable name="bpmsys"
select="//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']//*[name()='OBX.5']"/>
<xsl:variable name="bpmdia"
select="//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_DIA']//*[name()='OBX.5']"/>
<xsl:variable name="pulse"
select="//OBX[.//*[name()='CWE.2']='MDC_PULS_RATE_NON_INV']//*[name()='OBX.5']"/>
    <xsl:variable name="sms">
      We have received and processed your last bloodpressure measurement:
      Bloodpressure:<xsl:value-of select="$bpmsys"/>/<xsl:value-of select="$bpmdia"/>
      Pulse:<xsl:value-of select="$pulse"/>
    </xsl:variable>
    <reactionruleresult>
      <result>
        <xsl:copy-of select="SMS:CreateXMLForSMS($sms,
'46705619458','46705619458','reaction','r3@ct!0n')"/>
      </result>
    </reactionruleresult>
  </xsl:template>
<xsl:template match ="event[.//*[name()='CWE.2']='MDC_MASS_BODY_ACTUAL']">
<xsl:variable name="weight"
select="//OBX[.//*[name()='CWE.2']='MDC_MASS_BODY_ACTUAL']//*[name()='OBX.5']"/>
    <xsl:variable name="sms">
      We have received and processed your last weight measurement:
      Weight:<xsl:value-of select="$weight"/>
    </xsl:variable>
    <reactionruleresult>
      <result>
        <xsl:copy-of select="SMS:CreateXMLForSMS($sms,
'46705619458','46705619458','reaction','r3@ct!0n')"/>
      </result>
    </reactionruleresult>
  </xsl:template>
</xsl:stylesheet>
```

If we study one of the rules we see that it triggers (through the match attribute”) when we have received a blood pressure ORU-message (the code MDC_PRESS_BLD_NONINV_SYS).

```
<xsl:template match ="event[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']">
<xsl:variable name="bpmsys"
select="//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']//*[name()='OBX.5']"/>
<xsl:variable name="bpmdia"
select="//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_DIA']//*[name()='OBX.5']"/>
<xsl:variable name="pulse"
select="//OBX[.//*[name()='CWE.2']='MDC_PULS_RATE_NON_INV']//*[name()='OBX.5']"/>
    <xsl:variable name="sms">
      We have received and processed your last bloodpressure measurement:
      Bloodpressure:<xsl:value-of select="$bpmsys"/>/<xsl:value-of select="$bpmdia"/>
      Pulse:<xsl:value-of select="$pulse"/>
    </xsl:variable>
    <reactionruleresult>
      <result><xsl:copy-of select="SMS:CreateXMLForSMS($sms,
'46705619458','46705619458','reaction','r3@ct!0n')"/>
      </result>
    </reactionruleresult>
 </xsl:template>
```

Then there are 3 variable declarations that retrieves systolic, diastolic and pulse from the message XML.

```
<xsl:variable name="bpmsys"
select="//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']//*[name()='OBX.5']"/>
<xsl:variable name="bpmdia"
select="//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_DIA']//*[name()='OBX.5']"/>
<xsl:variable name="pulse"
select="//OBX[.//*[name()='CWE.2']='MDC_PULS_RATE_NON_INV']//*[name()='OBX.5']"/>
```

The variable “sms” then composes a message using the three vital signs values.

```
<xsl:variable name="sms">
      We have received and processed your last bloodpressure measurement:
      Bloodpressure:<xsl:value-of select="$bpmsys"/>/<xsl:value-of select="$bpmdia"/>
      Pulse:<xsl:value-of select="$pulse"/>
</xsl:variable>
```

Finally, the action taken is to use the “SMS”-object to send the message to a specified phonenumber.

```
<reactionruleresult>
      <result><xsl:copy-of select="SMS:CreateXMLForSMS($sms,
      '46705619458','46705619458','reaction','r3@ct!0n')"/>
      </result>
</reactionruleresult>
```

This object is an extension object to XSL-T, which allows developers to extend style sheets with their own functionality.

### 5.3 Orchestration Manager

The Orchestration Manager is responsible for managing and controlling execution of service orchestration schemes. A Service Orchestration is a high level description of how to execute a set of services in a specified sequence. The Service Orchestration is defined to support a specific workflow or task. The Service Orchestrations are defined by authorised stakeholders. The Orchestration Manager provides support for composite services and workflows. It is an execution engine for the REACTION Orchestration Language. Main functions are:

- Add orchestration scheme
- Update orchestration scheme
- Delete orchestration scheme
- Start orchestration

- Execute orchestration actions
- Stop orchestration
- Log and notify about orchestration events and actions
- Notify completion of orchestration
- Generate orchestration report

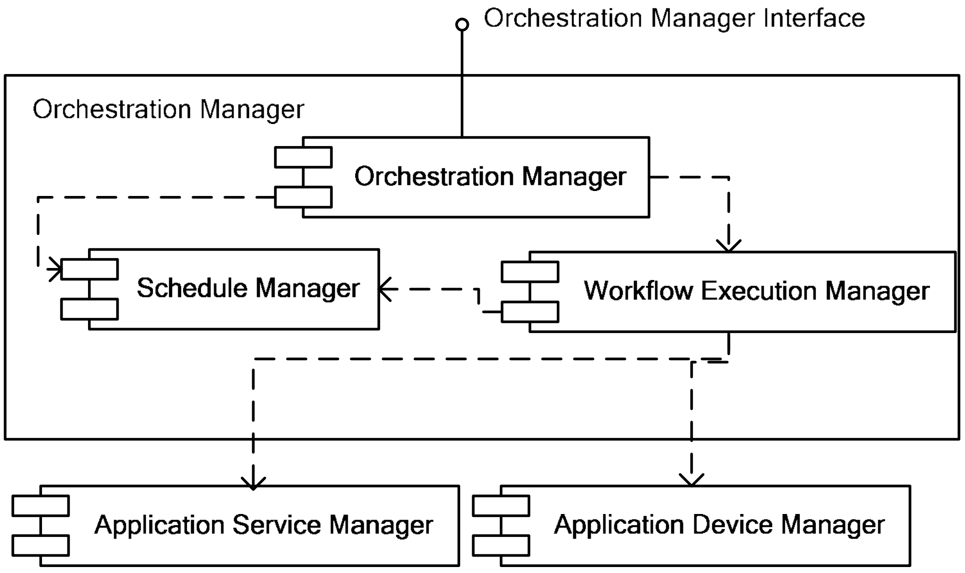

Orchestration Manager Interface
Orchestration Manager
Orchestration Manager
Schedule Manager
Workflow Execution Manager
Application Service Manager
Application Device Manager

Figure 7: Subcomponents of the Orchestration Manager.

**Schedule Manager**:

The scheduler is responsible for running tasks or notifying applications when a specific criterion is met. Such a criterion can be an e.g. specific (possibly recurring) time, system start-up, system shutdown.

**Workflow Execution Manager**:

The workflow execution module interprets orchestration descriptions and executes a set of services. These orchestrations may represent a complex service composed of other services or part of a REACTION application.

## 5.4 Alert and Alarm Manager

Purpose:
The Alert and Alarm Manager is responsible for deciding if an alert is to be generated or an alarm fired. It is informed of new patient events. Alerts are notifications that can be sent to physicians, informal carers or patients depending on the situation. The alerts can be sent using SMS, mail, or result in an update in a database. An Alarm is generated when a critical situation has occurred and requires immediate actions. The Alert and Alarm Manager uses the Rule engine to evaluate alert and alarm rules. Main functions are:

- Add alert rule
- Update  alert rule
- Delete alert rule
- Add alarm rule
- Update alarm rule
- Delete alarm rule
- Generate alert report

# 6 Network Management

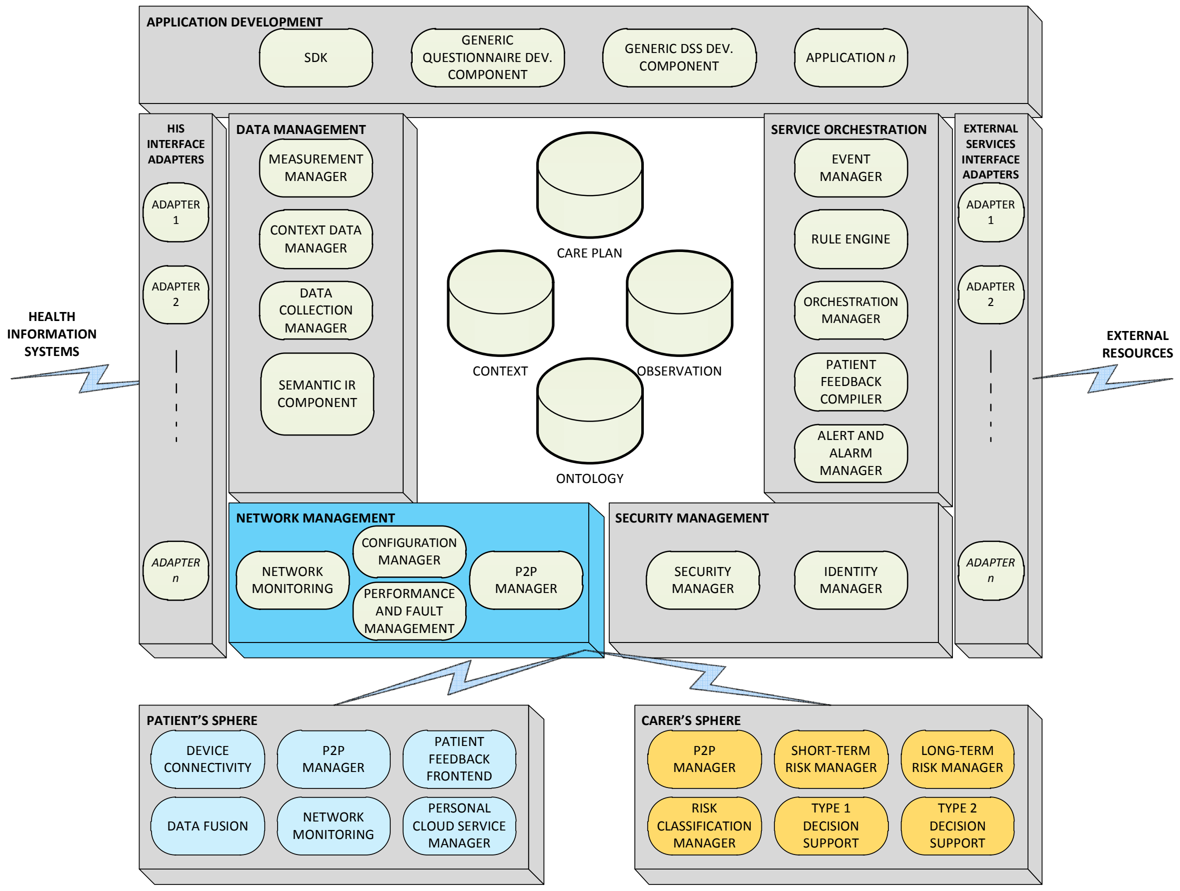


Figure 8: The Network Management subset and components.

## 6.1 P2P Manager

The Peer-to-Peer (P2P) Manager is the bottom layer of the LinkSmart middleware deployed in REACTION Gateways and in Internet-of-Things-enabled devices (IoT Devices). It is the entry and exit point of information of the LinkSmart middleware. There is only one Network Manager per device where the middleware is deployed.

The P2P Manager provides a Web Service interface (which is the main interface of the P2P Manager), which is the information entry point for the middleware. Data transferred between IoT-enabled devices and gateways should always pass through the Network Manager.

The P2P Manager is responsible of managing the communication between IoT -enabled devices. In order to do this, the P2P Manager:

- Creates and overlay P2P network, where all the IoT-enabled devices appear directly interconnected, no matter if they are behind a NAT (Network Address Translator) or Firewall.
- Provides indirection architecture for addressing Web Services hosted by IoT devices using the HID addressing mechanism. Each service is identified in IoT through an HID, which is a global and unique identifier. The P2P Manager provides interfaces for other managers, applications and IoT devices for HID creation, modification and deletion. It also offers the possibility to select the transport protocol for the service invocation between TCP, UDP and Bluetooth.
- Provides a transport mechanism over the overlay P2P network for invoking Web Services hosted by IoT devices (SOAP Tunnelling) using the HID addressing mechanism. The SOAP messages addressed to an HID are routed by the P2P Manager through the overlay network

to the P2P Manager hosting the service. Therefore, using the SOAP Tunnelling and the P2P Manager any device or application is able to transparently publish and consume services anywhere, anytime, breaking the network interconnectivity barriers and independently of the service endpoint location.

- Provides a transport mechanism over the overlay P2P network for multimedia content exchange between UPnP AV or DLNA devices.
- Provides session management mechanisms between HIDs during service invocations.
- Provides time reference synchronization between different P2P Managers.
- Provides a status page for developers, which the developer can use for monitoring dynamic information about the Internet-of-Things Network and the HIDs available.

Figure 9 presents the sub-managers included in the P2P Manager and their relationships.

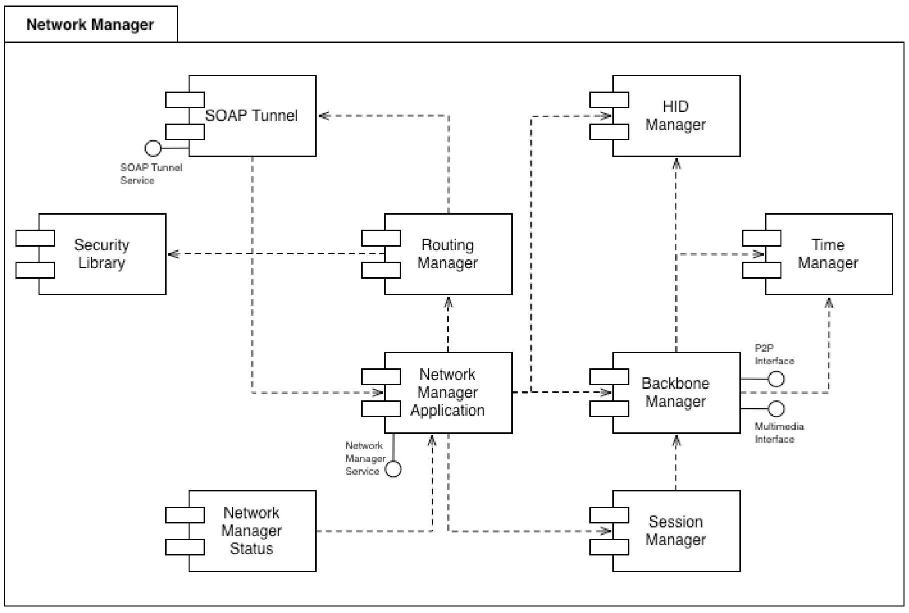


Figure 9: The P2P Manager Sub-Components.

## 6.2 Performance and Fault Management

The Network Monitoring component utilizes SNMP information to monitor the various AHDs in the REACTION network. SNMP is suitable for remote management, configuration and basic network monitoring, but less information can be retrieved to do further network traffic analysis. On the other hand, this type of information is very useful for further analysis. The purpose of the Performance and Fault Management component is to implement a network traffic probe to collect and analyse network traffic usage in order to track relevant network activities including network utilization, network protocol usage, traffic classification, fault detection, etc.

The basic functionality of the Performance and Fault Management subsystem can be summarized in the following main areas:

- Classify traffic and track network traffic usage
- Track network bandwidth utilization
- Identify performance and security issues

Figure 10 provides an overview on the various subsystems comprising the Performance and Fault Management component.

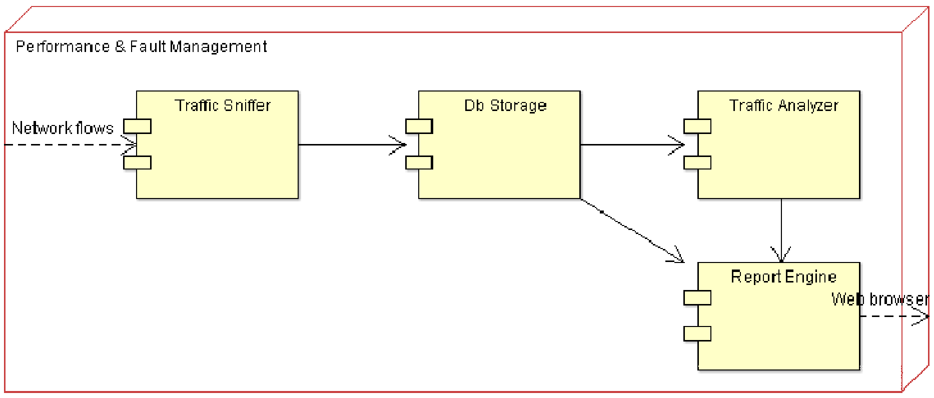


Figure 10: Performance and Fault Management architecture.

The Traffic Sniffer collects network packets and stores them on the database. This information is then passed to the Traffic Analyzer for processing. Whenever traffic information needs to be displayed the Report Engine renders the requested information appropriately. The following subsections further describe how the responsibilities are distributed between the internal subsystems of the Performance and Fault Management component.

### Traffic Sniffer

This component is responsible for collecting network flows from routers in the REACTION network and storing them for further analysis. The main functionalities of the Trap Dispatcher are the following:

- Filter network flows based on predefined preferences.
- Send data over the network using the NetFlow protocol
- Receive data from the network using the NetFlow protocol

### Database Storage

This component is responsible for persisting traffic flows information. The main functionalities of the Android Management Information Base (MIB) are the following:

- Create the schema for the traffic flows information.
- Store traffic flows to persistent storage.
- Retrieve traffic flows from persistent storage.

### Traffic Analyzer

This component is responsible for analyzing traffic information that has been retrieved by the Traffic Sniffer component. The analysis involves traffic classification, identification of specific network activities (e.g. evaluation of specified rules for certain network variables), etc. The main functionalities of the Trap Collector are the following:

- Debug network problems.
- Gather statistical data.
- Track suspicious access to specified network resources.

### Report Engine

This component is responsible for presenting the statistical information which is the output of the Traffic Analyzer component through a web interface. The Database Storage component has the following main functionalities:

- Generate reports allowing remote users and administrators to analyze traffic statistics by means of a web browser.

The Performance Fault Manager component does not depend on any other component in the REACTION backend for its proper operation. The Performance and Fault Manager component does not expose a public interface to other REACTION components. It makes available Performance and Fault monitoring information, in various forms, to network administrators through its Report Engine component, which is accessible through the relevant web URL.

## 6.3 Network Monitoring

The Network Monitoring subsystem monitors data traffic and assesses the transmission quality between the Patients' Sphere and the Carer's Sphere, the REACTION backend and other backend systems and EHRs with the Body Area Network (BAN) and Personal Area Network (PANS) components.

To enable network management functionality in all these areas of the REACTION system, a selection of monitoring points in the system must be made where appropriate software will be installed. This essentially constitutes the topology of the network management infrastructure.

Furthermore, the type of information to be collected from network elements must be specified to enable the monitoring framework to poll network elements for this information, process it and assess various aspects of the network including security, Quality of Service (QoS), availability, etc.

The basic functionality of the Network Monitoring subsystem can be summarized in three main areas:

- The first functional area involves the initialization of the android devices in the REACTION network. This entails an automatic configuration process which is controlled by the REACTION backend (an Edge Monitoring Node, or EMN, to be precise) and during which the device obtains information that is necessary to start communicating with the REACTION backend, e.g. a unique identifier, a description of the information to be periodically transmitted, etc.
- The second area involves the action of broadcasting status updates by the android devices operated either at the patient's premises or the hospital wards.
- The last functional area is the periodic polling from the monitoring server in order to collect the status for the various devices in the network. Each function includes complementary actions, such as retrieving the status of each device from the underlying OS, storing the status in a database at the monitoring server, etc.

To better understand the mentioned functionality an examination of the network monitoring topology is necessary. This is presented in the component diagram depicted in Figure 11. Figure 11Figure 11. Figure 11 provides an overview on the various subsystems comprising the Network Monitoring component.

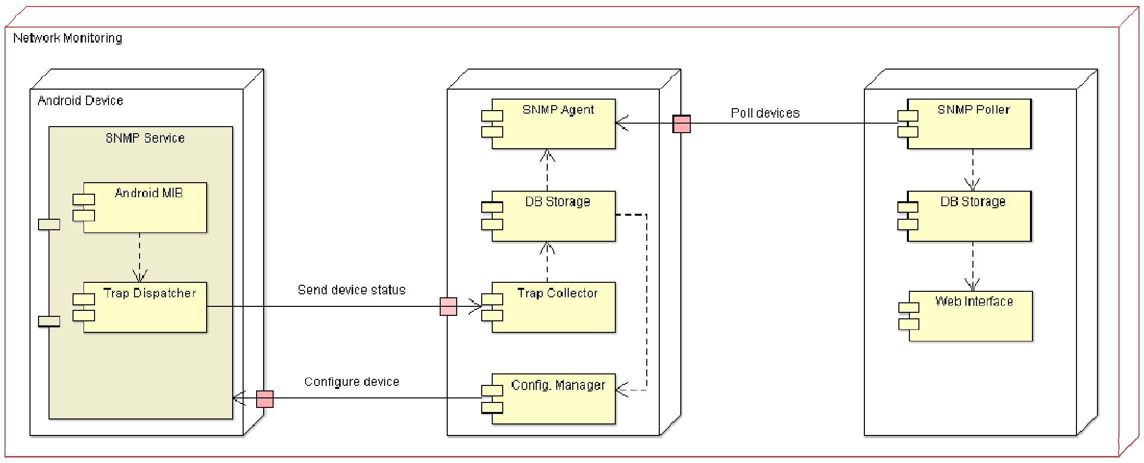


Figure 11: Network Monitoring topology.

A three-layer topology is shown in the diagram consisting of the following elements in each layer:

- Android devices (Application Hosting Devices - AHDs)
- Edge Monitoring Nodes (EMNs)
- A Central Monitoring Node (CMN)

The EMNs act as proxies for the various AHDs essentially serving as intermediate collection points of information regarding the status of the android devices. The CMN polls the EMNs for this information, as well as any other network device is considered necessary. The EMNs are deployed within separate organizational boundaries in order to collect status updates from the AHDs operated within the same organization (e.g. hospitals or primary care facilities).

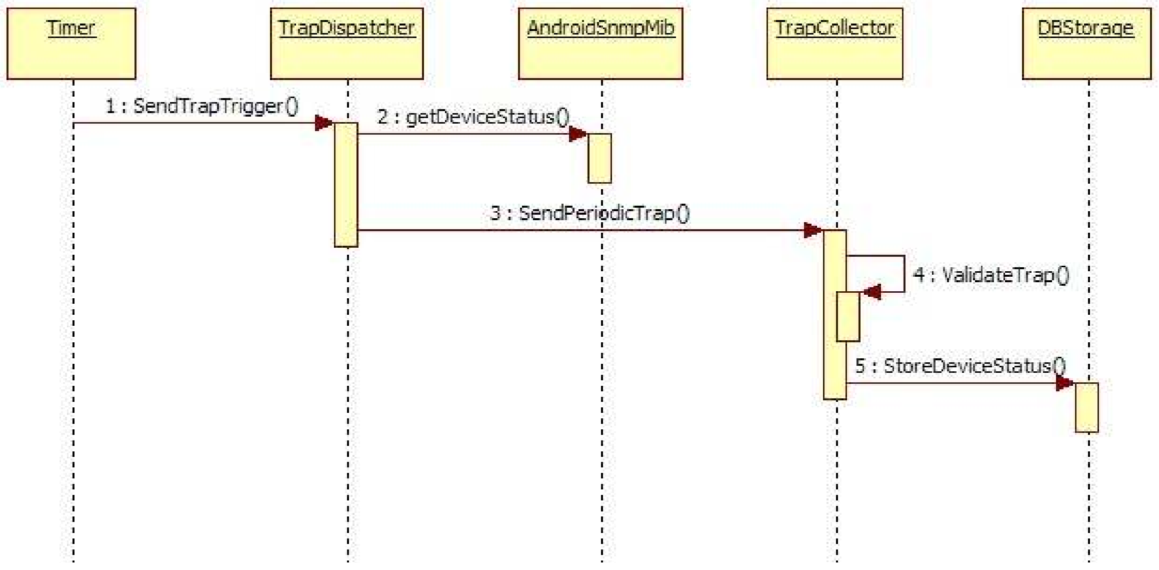
From the component diagram it is evident that there are three interaction points between the three tiers of the architecture. The first interaction involves the ConfigurationManager component on the EMNs with the SNMP Service on the AHDs. The second interaction involves the TrapDispatcher component on the AHDs with the TrapCollector component on the EMNs, while the third interaction is between the SNMPAgent component on the EMNs and the SNMPPoller component on the CMN. These interactions are independent of each other and constitute the main integration elements on which the integration tests need to be focused on.

The following subsections further describe how the responsibilities are distributed between the internal subsystems of the Network Monitoring component.

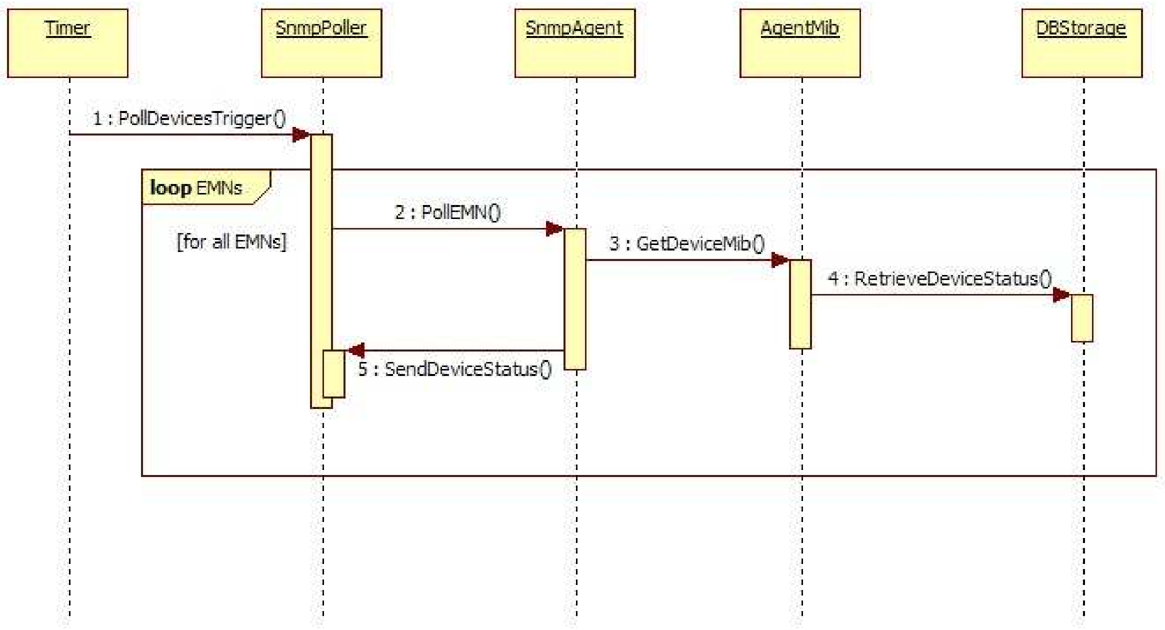
### Network monitoring subsystems' interactions

In this section the interactions between the three tiers comprising the Network Monitoring component are further analyzed through appropriate UML sequence diagrams to provide the details that are necessary in order to formulate the necessary integration tests.

In the first sequence diagram the interactions between the various components involved for sending periodic notifications from the AHDs to the associated EMN are depicted. This diagram captures the interactions that need to be tested to verify a successful integration between layer 1 (AHDs) and layer 2 (EMNs) of the network monitoring subsystem.



In the second sequence diagram the interaction between the CMN and the various EMNs is depicted. This diagram captures the interactions that need to be tested to verify a successful integration between layer 2 (EMNs) and layer 3 (CMN) of the network monitoring subsystem.



Beside the tests required to verify the interactions depicted in the two sequence diagrams, tests are also necessary to verify the end-to-end integration for the system. In other words verifying that Layer1-Layer2 and Layer2-Layer3 integration is successful does not guarantee end-to-end integration (Layer1-Layer2-Layer3).

**Trap Dispatcher (Android Device)**

This component is responsible for sending the periodic SNMP device status updates to the REACTION backend. The main functionalities of the Trap Dispatcher are the following:

- Create appropriate SNMP notifications.
- Send status update.

**Android MIB (Android Device)**

This component is responsible for describing the type of information to be included in the device status updates that are sent to the REACTION backend. The main functionalities of the Android MIB are the following:

- Add device status element.
- Get device status element.

**Trap Collector (EMN)**

This component is responsible for collecting and verifying the SNMP status updates from all registered devices residing in the associated organizational subnet. The main functionalities of the Trap Collector are the following:

- Receive new status update from registered device.
- Verify SNMP status update.

**Database Storage (EMN)**

This component is responsible for storing the received status updates to persistent storage in such a way that they can be easily polled by the CMN. The Database Storage component has the following main functionalities:

- Define/Create status update database schema.
- Store new status update.

**SNMP Agent (EMN)**

This component is responsible for answering to polling requests by the CMN and retrieving status updates from persistent storage. The SNMP Agent component has the following main functionalities:

- Handle SNMP polling request.
- Retrieve status updates from the Db Storage component.
- Create SNMP response with requested information.
- Send SNMP reply to poller.

**Configuration Manager (EMN)**

This component is responsible for registering and unregistering REACTION devices and providing the necessary information so that each device can be initialized and start sending periodic notifications. The Configuration Manager component has the following main functionalities:

- Register new device.
- Un-register device.
- Send configuration information.

**Web Interface (CMN)**

This component is responsible for presenting information regarding the network status of each device in the REACTION network to administrative personnel. It also provides notification options for alerting the appropriate personnel when predefined network conditions occur. The Web Interface utilizes the Cacti open source NMS to implement the necessary functionality. The Web Interface component has the following main functionalities:

- Present network monitoring information in appropriate form.
- Provide notification alerts.
- Register/unregister EMNs to be monitored.
- Manage polling intervals for the various monitored devices.
- Provide network templates to specify and manage the network variables to be monitored.

**Database Storage (CMN)**

This component is responsible for storing the obtained monitoring information from each registered EMN to persistent storage in such a way that they can be easily polled by the CMN. The Database Storage component has the following main functionalities:

- Define/Create network monitoring db schema.
- Store monitoring information.

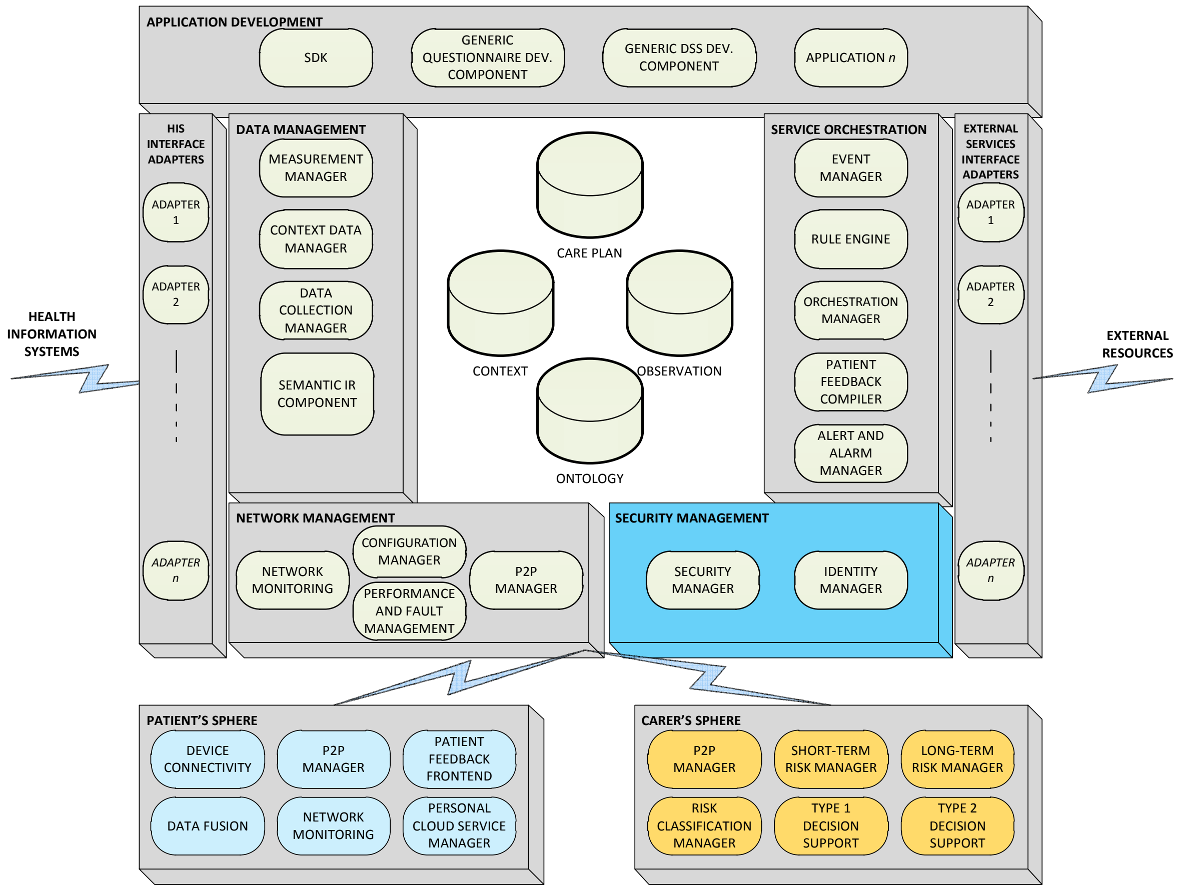
**SNMP Poller (EMN)**

This component is responsible for polling the registered EMNs at the specified polling intervals for the network variables defined at the appropriate network templates. The SNMP Poller component has the following main functionalities:

- Poll registered EMNs to store the current network status.

The Network Monitoring component does not expose a public interface to other REACTION components. It makes available network monitoring information, in various forms, to network administrators through its Web Interface component, which is accessible through the relevant web URL.

# 7 Security Management



## 7.1 Security Manager

The Security Manager is responsible for providing authenticated and/or confidential communication between the REACTION system and its users, which will be the meaning of ‘security’ for the purpose of this section. The Security Manager will also control access to the components described in this document, i.e., it will only permit access to the Web services implemented by the components to authorised parties. The purpose of the Security Manager is to provide basic security for all components and not application or service specific security. If necessary, the latter needs to be implemented separately.

Unlike other components specified in this document, the Security Manager is not a Web service and operates at a level ‘below’ other components. A rough sketch of a typical ‘Web service stack’ is shown in Figure 12.

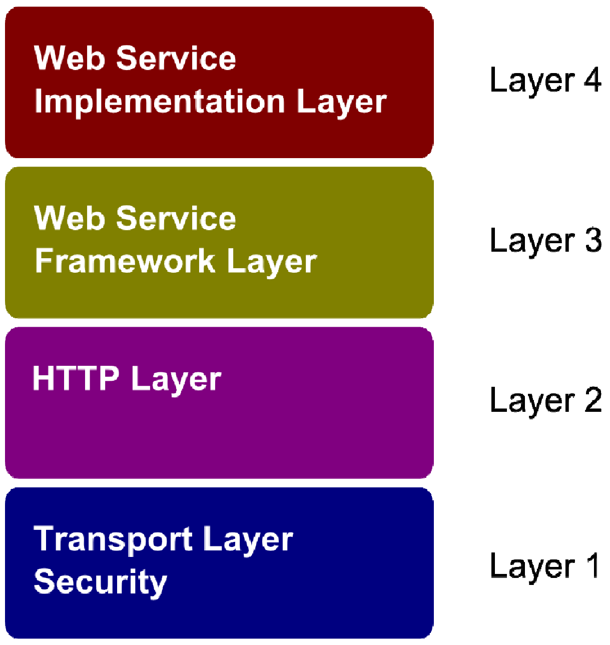


Figure 12: Web service stack.

Components described throughout this document are typically located at the Web Service Implementation Layer, i.e., layer 4, and implement an interface imposed by the API of the underlying Web Service Framework located at layer 3. This framework, in turn, will run on a Web server which handles the basic communication, i.e., the transport of the Web service messages at layer 2. In addition, Web servers typically also support security at layer 1, meaning that the communication between the server and its clients is encrypted and, at least, the server can authenticate to its clients.

In principle, security can be provided at any layer mentioned before. However, depending on the layer where security is implemented, the impact on the components differs. In the following, we will briefly go through the implications of having security implementations at different layers of the stack. Note that the following discussion only pertains to the server side. As a second note, the options discussed below, except for the layer 4 option, aim at providing security in a transparent way, i.e., component developers should not need to care about security.

Implementing security at layer 4 means that every component developer would have to take care of security by himself, e.g., by encrypting results returned by the component. This is not only error prone, since developers may leave out encryption on purpose or simply forget to encrypt the results before returning them, but the results can also be difficult to interpret by users of the service since they need to know how the results where encrypted in order to be able to decrypt them. Hence, implementing security at layer 4 does not seem to be an option.

Implementing security at layer 3 means that a security module will have to be created for the concrete Web Service Framework at hand, e.g., Axis, Axis2, CXF (Crossfire). This means, the module will have to be tailored to a specific framework and it is unlikely to be usable for any other framework. Hence for the development of the components, it is imperative to commit to a single framework as the work for supporting multiple frameworks is most likely linear in the number of frameworks and thus unviable. On the positive side, at this layer standards exist, like Web Service Security (WSS), which can be employed to allow for somewhat uniform handling of secured Web service messages. “Somewhat” means that while WSS defines syntax for describing signed and/or encrypted content, it still has many options which need to be determined beforehand. Also if different frameworks come with their own WSS implementations, their processing algorithms may differ and thus, uniform handling is not guaranteed. In conclusion, if components are implemented using different Web service frameworks, implementing security at layer 3 is *not* an option.

Implementing security at layer 2 means to directly operate on the ‘raw’ HTTP messages containing the Web service messages. This could be done, e.g., by tunnelling the HTTP messages through a transparent Web service proxy. A security proxy essentially works like the security module described for layer 3, i.e., it could use WSS to provide security. The major difference between a security proxy and a framework security module is that the proxy would be a stand-alone component with a separate stack, e.g., it may not even require layers 3 and 4, and would be independent of any framework. Thus, the framework limitation coming from a layer 3 implementation would be gone. However, since the proxy has its own stack, it has to forward its output, e.g., the decrypted Web service message, to another server which hosts the service component for which the Web service message is intended. Since this server would not implement any security, the message is 'unprotected' when travelling from the proxy to the component's host. Depending on the trust model, this may or may not be a security problem. If the security proxy would be running on the same host as the server hosting the Web service component and the trust model is such that communication on the same device is assumed to be trusted, then no security problem arises – this is the trust model used for the in-hospital prototype. Likewise, if the trust model would be such that services running in the same network[1] are trusted, the security proxy could be run on a different device and the 'unprotected' network communication between the proxy's host and the service component's host would not be a security problem, too. However, if a proxy cannot run on the same host as the Web service component and it cannot be assumed that the proxy's host and the Web service component's host are running in a trusted network then implementing security at layer 2 is *not* an option.

Implementing security at layer 1 means that HTTP messages are tunnelled through a security protocol, namely the Transport Layer Security (TLS) protocol. From an implementation point of view, having security at layer 1 is comparable to having security at layer 3. Since TLS is normally provided by the Web server, a security component controlling a Web server's TLS engine will have to be tailored to a specific Web server type, e.g., Apache Tomcat 6. On the positive side, having security at layer 1 can be more efficient than having it at other layers because security negotiations are made before potentially large Web service messages are sent and received. In addition, although a security engine at layer 1 depends on the concrete Web server, it may be independent of the Web service framework, provided that the different frameworks can be run by the same Web server type. In conclusion, if components require different server types then implementing security at layer 1 is *not* an option.

Main functions are:

- Authenticate users of components
- Decrypt incoming messages / encrypt outgoing messages, if necessary
- Control access to components

The following are components which would be required to implement security dependent on the option chosen. In any case, only one out of the four options will be used.

- Layer 4 option – Library (Jar)
- Layer 3 option – Framework Module
- Layer 2 option – Web Service Proxy
- Layer 1 option – TLS engine

The following list outlines who/what would be constrained if security is to be implemented at the given layer.

- Layer 4 option – Developer of Component
- Layer 3 option – Web Service Framework being used
- Layer 2 option – Trust Model employed for component hosts
- Layer 1 option – Web Server type

## 7.2 Identity Manager

The purpose of the Identity Manager is to support an Identity Security infrastructure (with minimum overhead) which is required to block actions from identities. The security and privacy chain as well as important trust concerns will therefore be identified and ranked so that secure identity management can be implemented throughout the platform. Hence the REACTION project aims to provide a visible

[1] 'Same network does not necessarily mean 'physically connected' hosts but could also include different physical networks connected through a VPN.

and controllable distributed security and privacy model, which is based on the concept of trust as a multilateral relation between stakeholders in a community of patients, informal carers and healthcare professionals and providers.

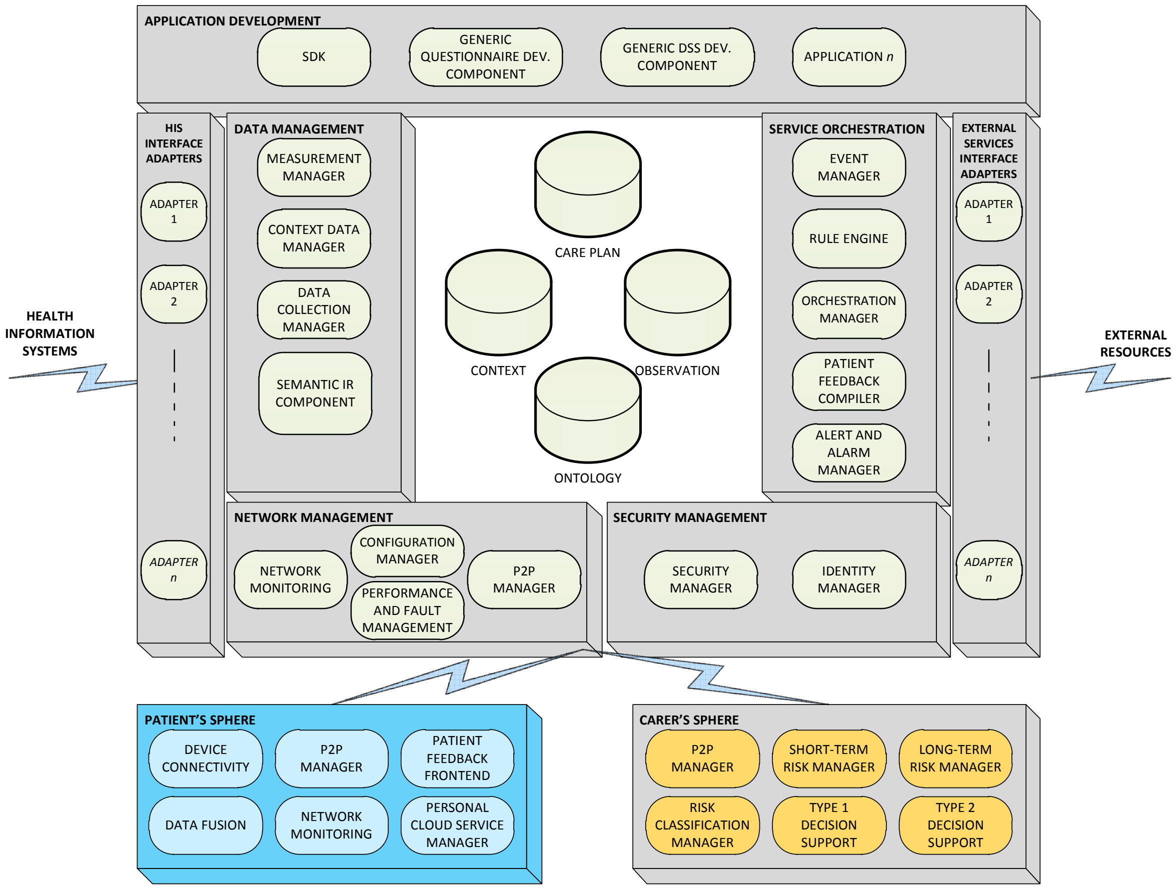
In sum, REACTION will combine identity virtualisation with infrastructure-based accountability negotiations, which are very efficient but also complex.

Main functions are:

- Create new user.
- Update user.
- Delete user.
- Assign role.
- Add password.
- Change password.

# 8 REACTION Device Connectivity Kit

The REACTION Device Connectivity Kit is used to implement solutions for integrating and interfacing with various medical device types. Normal it used to create the software that runs in the Patient Sphere, for instance on a patient home gateway.



## 8.1 Overview of the Device Connectivity Kit

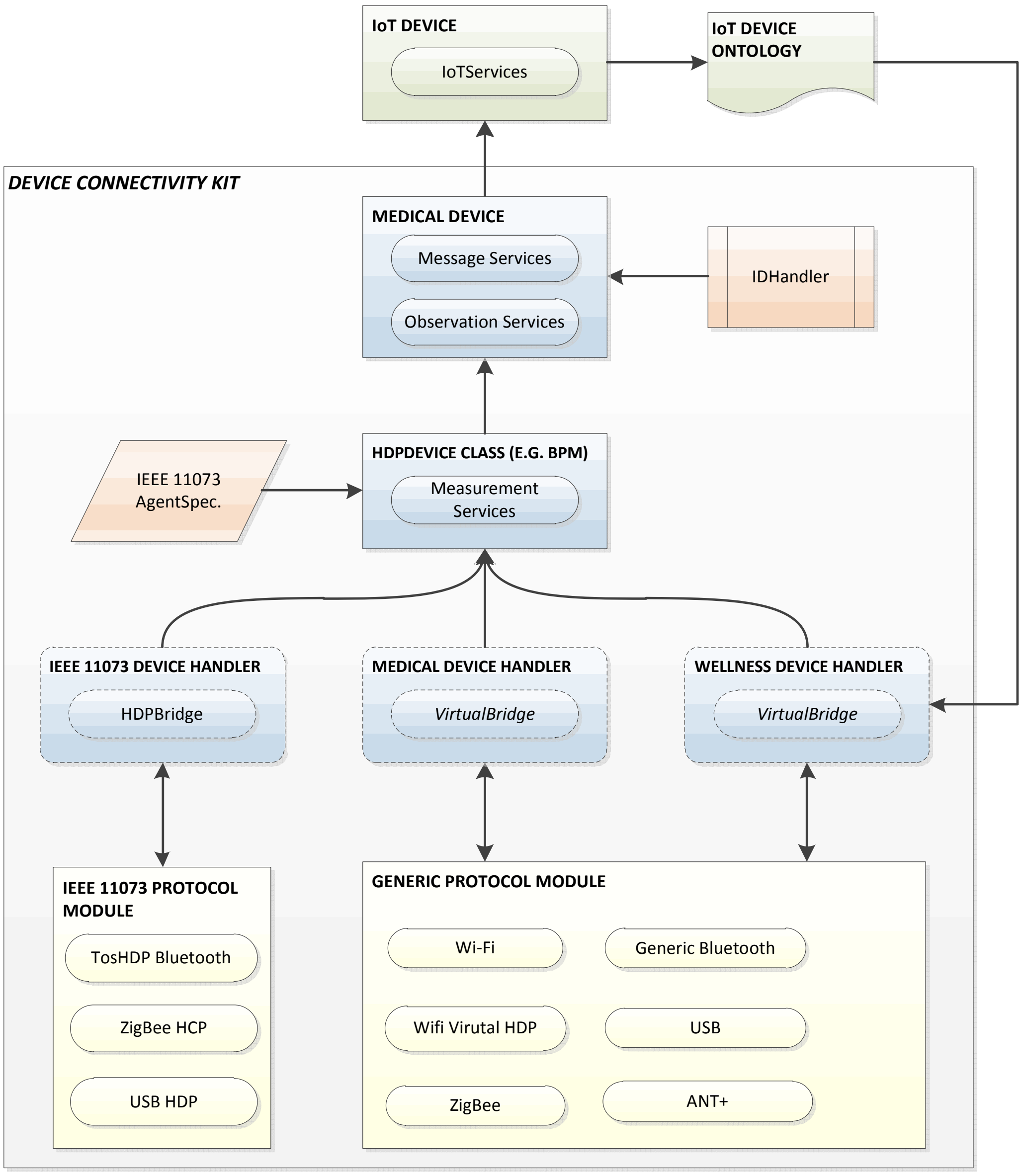


Figure 13: Hierarchical architecture of the classes used in the DCK.

## 8.2 IoT Device

IoT Device is the base class for all devices and sensors that have been developed using LinkSmart in an Internet of Things (IoT) application. It exposes a generic IoT Service. The IoT Device Manager handles several service requests and manages the responses. The IoT  Device Manager class is a generic class that is sub-classed depending on device type, example of subclasses are Medical Device Manager, BloodPressureMonitor Device Manager. Bluetooth Device Manager, Basic Phone Device Manager, Basic Switch Device Manager.

Main Functions are:

- Maps requests to device services
- Generates responses
- Advertising IoT device descriptions
- Advertises device services

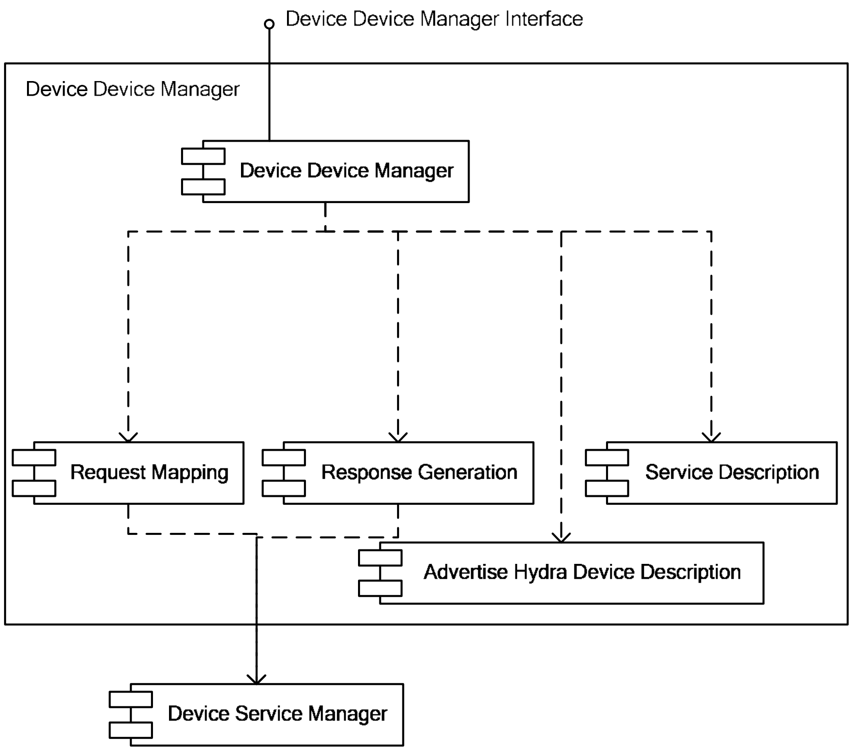


Figure 14: IoT Device Manager

Advertise:
This module is responsible for broadcasting the existence of the device to the outside world. It will support advertising thru several protocols, at least UPnP (Universal Plug and Play).

Request Mapping:
This module maps a request from an outside caller to an internal service in the device.

Response Generator:
This module maps translates the result of an internal service in the device to a response to the caller.

## 8.3 Medical Device

Medical Device is the base class for all medical devices and sensors that have been integrated using the Device Connectivity Kit. The Medical Device Manager is a subclass of the IoT Device Manager. It exposes a generic Medical Service and can push measurements to the server side, and also supports on-demand retrieval of data from the patient side.

Main functions are receive measurement from physical device, send measurement to server side or other modules, receive events from physical device, propagate events to Event Manager and get observation as plain ORU or XML ORU.

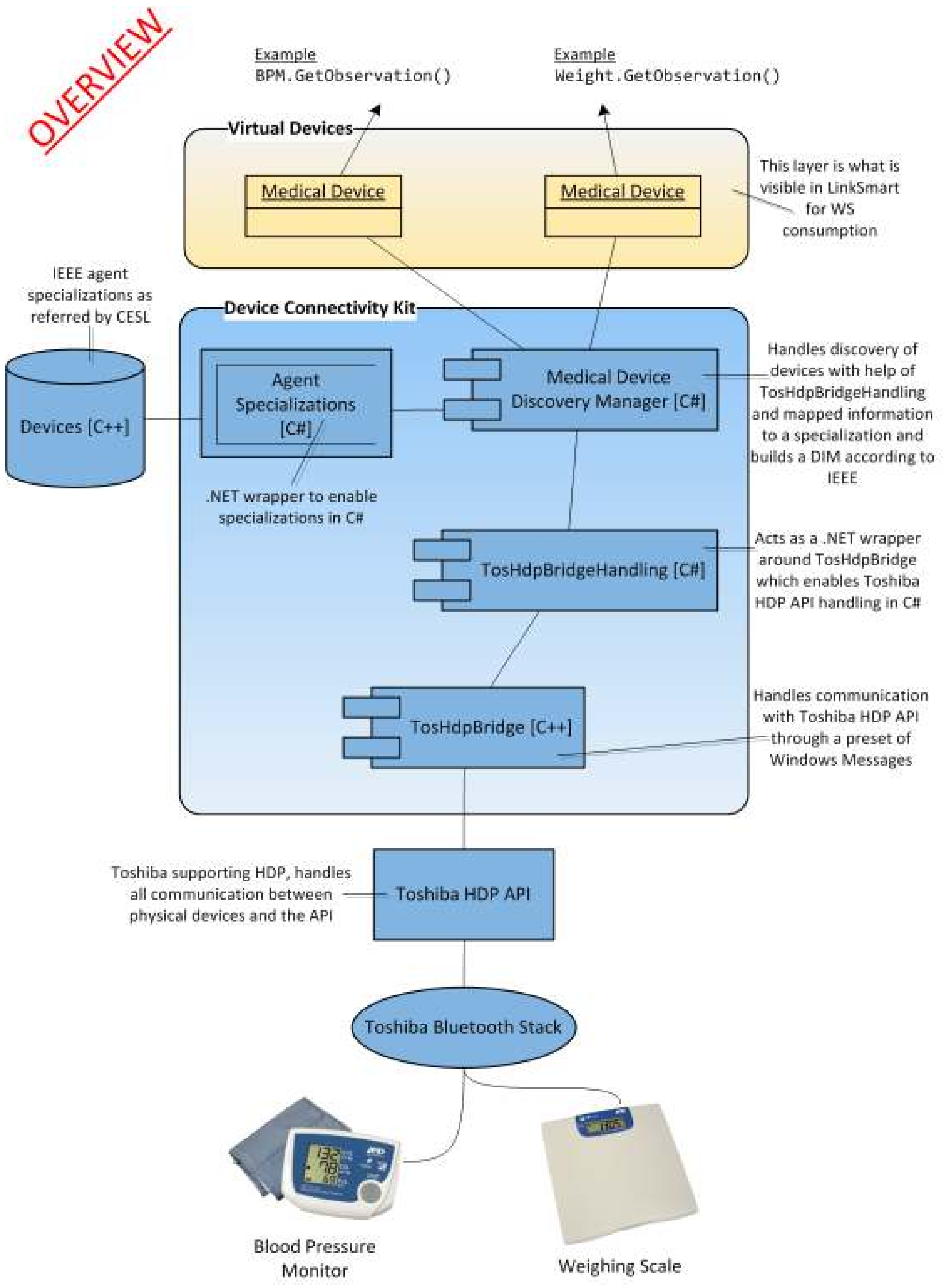

OVERVIEW
Example
BPM.GetObservation()
Example
Weight.GetObservation()
Virtual Devices
Medical Device
Medical Device
This layer is what is visible in LinkSmart for WS consumption
IEEE agent specializations as referred by CESL
Device Connectivity Kit
Devices [C++]
Agent Specializations [C#]
Medical Device Discovery Manager [C#]
Handles discovery of devices with help of TosHdpBridgeHandling and mapped information to a specialization and builds a DIM according to IEEE
.NET wrapper to enable specializations in C#
TosHdpBridgeHandling [C#]
Acts as a .NET wrapper around TosHdpBridge which enables Toshiba HDP API handling in C#
TosHdpBridge [C++]
Handles communication with Toshiba HDP API through a preset of Windows Messages
Toshiba supporting HDP, handles all communication between physical devices and the API
Toshiba HDP API
Toshiba Bluetooth Stack
Blood Pressure Monitor
Weighing Scale

Figure 15 Overview of Medical Device components.

## 8.4 REACTION Test Suite

The Test Suite is a tool that can be used to create Medical Devices from the Device Connectivity Kit, and supports Blood Pressure Monitor, Weighing Scale, Glucose Meter and Pulse Oximeter. The communication to the device is handled through the Medical Device Manager.

Main Functions are:

- Create Medical Devices
- Set a new measurement
- Invoke Medical Device to send a HL7 ORUR01 message
- Test the Medical Device software functionality
- Test server side components that handle

The Test Suite is a Silverlight application that uses the Test Suite Handler built into the Device Connectivity Kit.

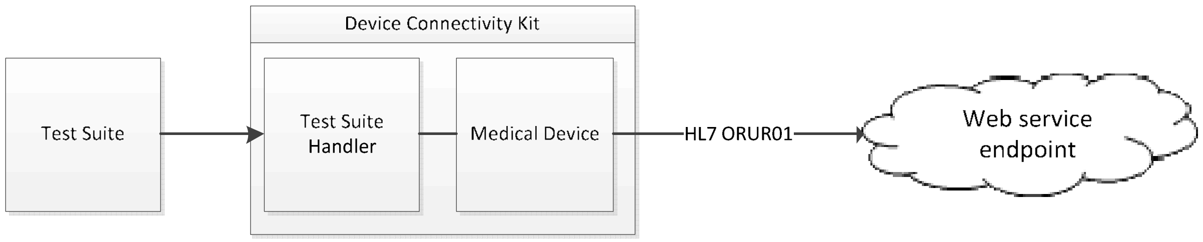
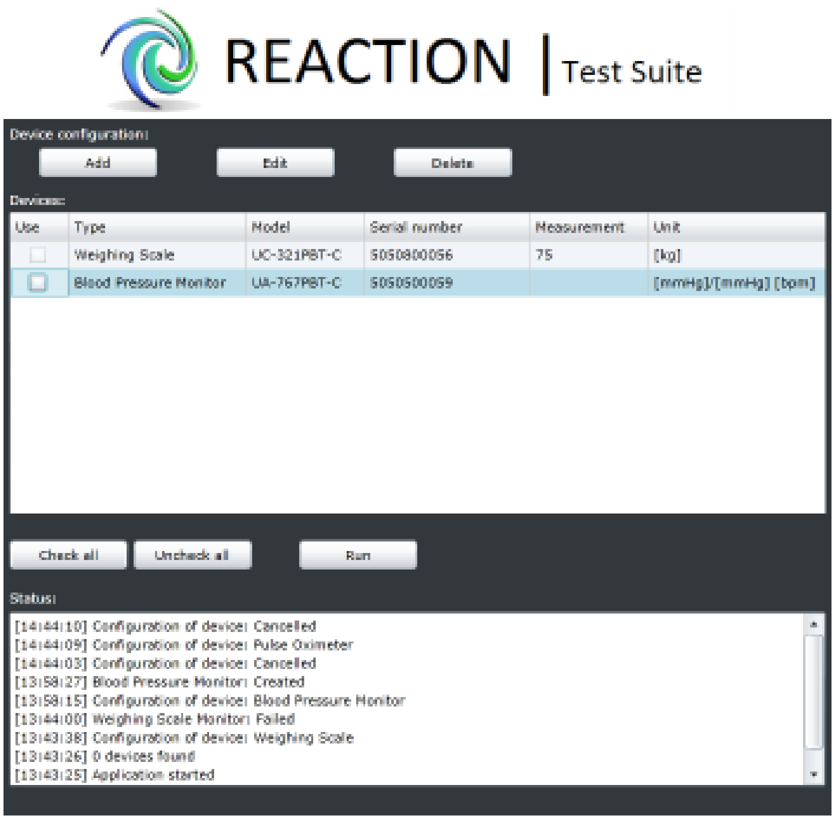


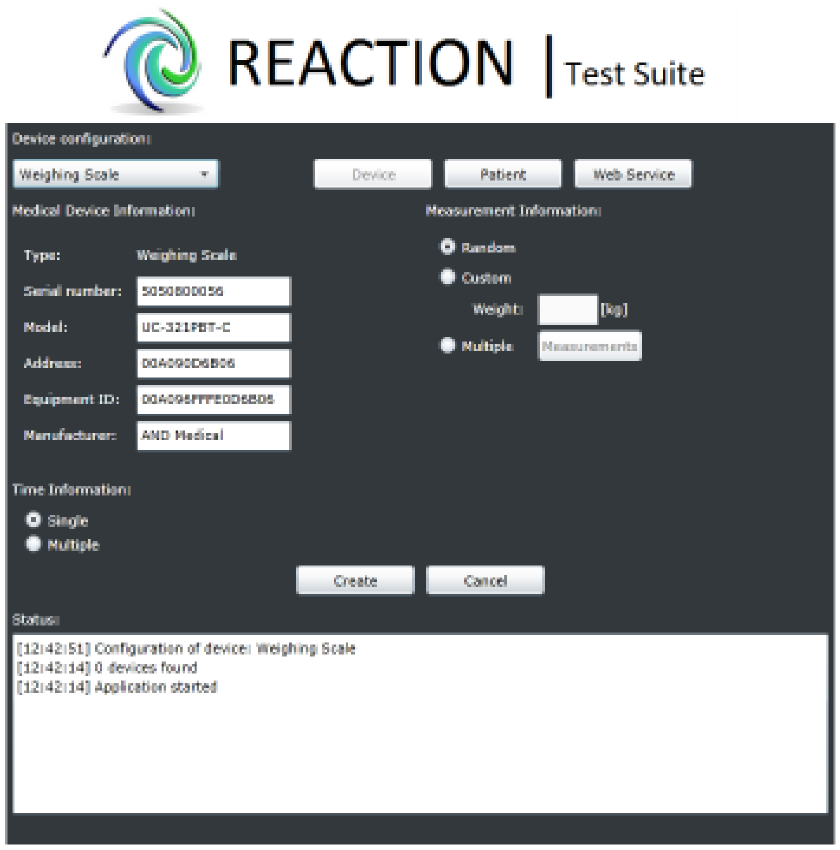
Figure 16: Test Suite component overview.



| Use | Type | Model | Serial number | Measurement | Unit |
|---|---|---|---|---|---|
| ☐ | Weighing Scale | UC-321PBT-C | 5050800056 | 75 | [kg] |
| ☐ | Blood Pressure Monitor | UA-767PBT-C | 5050500059 | | [mmHg]/[mmHg] [bpm] |

```
[14:44:10] Configuration of device: Cancelled
[14:44:09] Configuration of device: Pulse Oximeter
[14:44:03] Configuration of device: Cancelled
[13:58:27] Blood Pressure Monitor: Created
[13:58:15] Configuration of device: Blood Pressure Monitor
[13:44:00] Weighing Scale Monitor: Failed
[13:43:38] Configuration of device: Weighing Scale
[13:43:26] 0 devices found
[13:43:25] Application started
```

Figure 17: Main page and overview of created devices.



Type: Weighing Scale
Serial number: 5050800056
Model: UC-321PBT-C
Address: 00A090D6B06
Equipment ID: 00A096FFFE0D6B06
Manufacturer: AND Medical

Measurement Information: [✓ Random] Custom Multiple

Time Information: [✓ Single] Multiple

```
[12:42:51] Configuration of device: Weighing Scale
[12:42:14] 0 devices found
[12:42:14] Application started
```

Figure 18: Configuration of a Weighing Scale device.

## 8.5 Data Fusion Engine

The Data Fusion engine is responsible for collecting, combining and aggregation data from two or more medical sensors/devices. In the Data Management subset, the Data Fusion engine will result from the focus on the design of a device and network-based data fusion/diffusion model. This model will provide a semantic integration of a multitude of heterogeneous medical devices and media, information sources and services and communication.

REACTION will primarily have to manage patient specific data fusion at a device level data fusion. This means that each patient is unique and therefore clinicians must be able to configure and control the medical measurements and the context data required for each separate patient under monitoring. To achieve this, the device level data fusion will need to capture relevant context data at the point of measurement and through history. The Data Fusion Engine will therefore have to handle different data values (time, temperature, location, etc.) from multiple devices and aggregate these into a single observation.

The work on the REACTION Data Fusion for the Client side is a long process and is currently covered by the ORU-R01 message segmentation while the Data Fusion Engine itself is manifested in the REACTION Server side and next coming section here.

The Data Fusion Engine subsystem takes part of the Data Management subset which is central to the high level functioning of applications and services deployed on the platform. In summary, the intention of the Data Fusion Engine is to 1) aggregate multiple data sources into uniform representation and 2) manage data transfer to and from nodes and stakeholders in a REACTION environment.

Main functions are:

- Set data fusion scheme
- Discovery and connect to sub devices
- Collect data
- Listen to device events

- Aggregate and correlate events and observations based on data fusion scheme.

The Data Fusion Engine in this second prototype is implemented as an IoT-enabled device using the Hydra middleware. This means it is possible to have a number of "virtual devices" doing specific data fusion tasks.

Such a data fusion device is configured using two configuration files. The first file defines the sub device which delivers the data to be fused by the Data Fusion Device. For instance the following file defines a data fusion device that will work with a weight scale and a blood pressure monitor..

```
<?xml version="1.0" encoding="UTF-8"?>
<datafusion>
       <subdevice xpath="//*[name()='friendlyName' and .='Medical Device: Weighing Scale
       Device']" name="WeigthScale"/>
       <subdevice xpath="//*[name()='friendlyName' and .='Medical Device: Blood Pressure
       Device']" name="BPM"/>
</datafusion>
```

The format is simple. The subdevice element of the XML file defines a data providing device. The xpath attribute is the search string that will select the device from the Application Device Catalogue in REACTION. Currently one limitation is that the xpath expression should result in one device, not a set of devices. The name attribute defines the name that the device will be known under and referred to in the fusion schemes, see details below.

The second file defines the algorithms and how to merge, fuse and integrate data. This file is called the fusion scheme. Currently it is expressed using XSL-T. The following example shows how weight and bloodpressure is merged together to one value:

```
<xsl:template match ="weightbpmvalue">
    <xsl:variable name="obx1" select="WeigthScale:ExecuteXML('urn:upnp-
    org:serviceId:medicalobservationservice:1','GetObservation','')"/>
    <xsl:variable name="obx2" select="bpm:ExecuteXML('urn:upnp-
    org:serviceId:medicalobservationservice:1','GetObservation','')"/>
    <xsl:variable name="weight" select="msxsl:node-
    set($obx1)//OBX[.//*[name()='CWE.2']='MDC_MASS_BODY_ACTUAL']//*[name()='OBX.5']"/>
    <xsl:variable name="bpmsys" select="msxsl:node-
    set($obx2)//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']//*[name()='OBX.5']"/>
    <xsl:variable name="bpmdia" select="msxsl:node-
    set($obx2)//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_DIA']//*[name()='OBX.5']"/>
    <xsl:variable name="pulse" select="msxsl:node-
    set($obx2)//OBX[.//*[name()='CWE.2']='MDC_PULS_RATE_NON_INV']//*[name()='OBX.5']"/>
    <fusedvalue>
      <weight>
        <xsl:value-of select="$weight"/>
      </weight>
      <bpmsys>
        <xsl:value-of select="$bpmsys"/>
      </bpmsys>
      <bpmdia>
        <xsl:value-of select="$bpmdia"/>
      </bpmdia>
      <pulse>
        <xsl:value-of select="$pulse"/>
      </pulse>
    </fusedvalue>
  </xsl:template>
```

The two first variable statements:

```
    <xsl:variable name="obx1" select="WeigthScale:ExecuteXML('urn:upnp-
    org:serviceId:medicalobservationservice:1','GetObservation','')"/>
    <xsl:variable name="obx2" select="bpm:ExecuteXML('urn:upnp-
    org:serviceId:medicalobservationservice:1','GetObservation','')"/>
```

These statements retrieve two ORU-messages from two different medical devices. The following statement simply extracts the different vital signs, by searching inside the ORU-message XML.

```
    <xsl:variable name="weight" select="msxsl:node-
    set($obx1)//OBX[.//*[name()='CWE.2']='MDC_MASS_BODY_ACTUAL']//*[name()='OBX.5']"/>
    <xsl:variable name="bpmsys" select="msxsl:node-
    set($obx1)//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']//*[name()='OBX.5']"/>
    <xsl:variable name="bpmdia" select="msxsl:node-
    set($obx1)//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_DIA']//*[name()='OBX.5']"/>
    <xsl:variable name="pulse" select="msxsl:node-
    set($obx1)//OBX[.//*[name()='CWE.2']='MDC_PULS_RATE_NON_INV']//*[name()='OBX.5']"/>
```

Finally, the following statements define how the different values should be assembled into one XML-structure and returned from the Data Fusion device:

```
<fusedvalue>
      <weight>
        <xsl:value-of select="$weight"/>
      </weight>
      <bpmsys>
        <xsl:value-of select="$bpmsys"/>
      </bpmsys>
      <bpmdia>
        <xsl:value-of select="$bpmdia"/>
      </bpmdia>
      <pulse>
        <xsl:value-of select="$pulse"/>
      </pulse>
</fusedvalue>
```

The result will then be and XML string looking like:

```
<fusedvalue>
      <weight>80</weight>
      <bpmsys>120</bpmsys>
      <bpmdia>70</bpmdia>
      <pulse>54</pulse>
</fusedvalue>
```

The XSL-template above is triggered when the method GetFusedValue(string name) is called from an external component. In this case the “name” parameter will trigger the correct template. In the example above, the call GetFusedValue(“weightbpmvalue”) should be used.

There is also the option to do the data fusion based on events occurring. The template below shows an example of this:

```
<xsl:template match ="topic[.='deviceStateChanged']">
       <xsl:variable name="obx1" select="WeigthScale:ExecuteXML('urn:upnp-
       org:serviceId:medicalobservationservice:1','GetObservation','')"/>
       <xsl:variable name="obx2" select="BPM:ExecuteXML('urn:upnp-
       org:serviceId:medicalobservationservice:1','GetObservation','')"/>
       <xsl:variable name="weight" select="msxsl:node-
       set($obx1)//OBX[.//*[name()='CWE.2']='MDC_MASS_BODY_ACTUAL']//*[name()='OBX.5']"/>
       <xsl:variable name="bpmsys" select="msxsl:node-
       set($obx2)//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_SYS']//*[name()='OBX.5']"
       />
       <xsl:variable name="bpmdia" select="msxsl:node-
       set($obx2)//OBX[.//*[name()='CWE.2']='MDC_PRESS_BLD_NONINV_DIA']//*[name()='OBX.5']"
       />
       <xsl:variable name="pulse" select="msxsl:node-
       set($obx2)//OBX[.//*[name()='CWE.2']='MDC_PULS_RATE_NON_INV']//*[name()='OBX.5']"/>
       <xsl:variable name="activity" select="ActivityHub:Execute('urn:upnp-
       org:serviceId:activityhubservice:1','GetLatestMeasurement','')"/>
       <xsl:variable name="motion">
       There has been movement in:
       <xsl:for-each select="msxsl:node-set($activity)//Sensor">
```

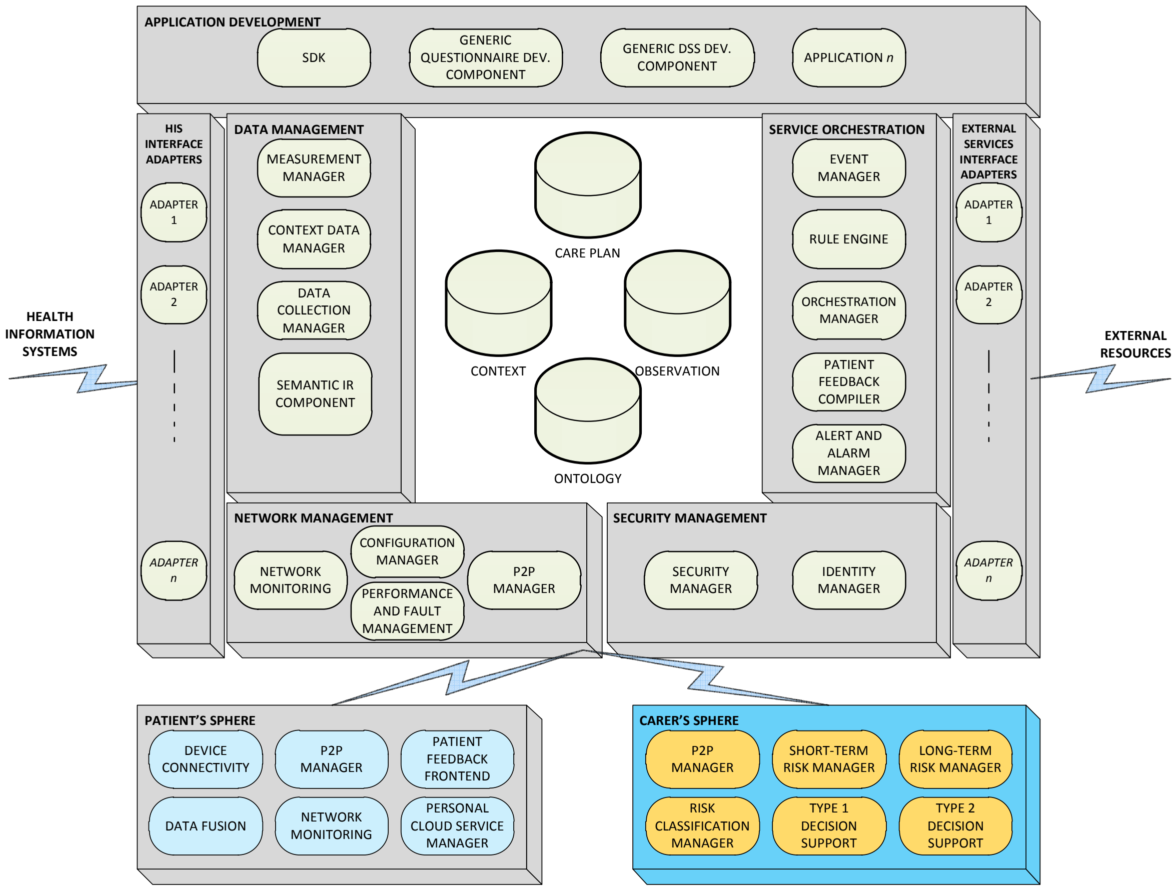
```
              <xsl:if test="Type ='MotionDetector' and Value='YES'">
              <xsl:value-of select="Location"/><xsl:if test="not(Sensor[last()])">,
              </xsl:if>
              </xsl:if>
       </xsl:for-each>
       </xsl:variable>
<xsl:variable name="fusionreport">
Datafusion report:
weight : <xsl:value-of select="$weight"/>
bloodpressure: <xsl:value-of select="$bpmsys"/>/<xsl:value-of select="$bpmdia"/>
pulse : <xsl:value-of select="$pulse"/>
LightSwitch Status: <xsl:value-of select="//eventpart[key='IoTVariableValue']/value"/>
Activity:<xsl:value-of select="$activity"/>
</xsl:variable>
<xsl:copy-of select="EventManager:Publish(newfusionvalue, fusionreport)"/>
</xsl:template>
```

This code assumes that two more sub devices – Activity Hub and a Lightswitch have been added to the definition of the Data Fusion Device. The XSL-template is triggered by an event that is generated when the lightswitch is turned on and off. This generates the event “devicestatechanged”. The code shows how the weight and blood pressure are collected. In addition to this the code traverses all the sensors attached to the activity hub to see if there has been any activity in different rooms. All the information is compiled into a “fusionreport”.

Then this fusionreport is published as an event using the REACTION Event Manager. In this way the data fusion engine acts as a high level layer so that other managers and applications don´t have to listen for low level events but can process aggregated information.

The configuration file and the data fusion schemes are in this prototype produced by the developer. Next step is to develop a graphical user interface that makes it simple to define different data fusion schemes.

# 9 Risk Management and Decision Support



## 9.1 Long-term Risk Manager

The Long Term Risk Assessment models (LTRAMs) component is designed to manage the REACTION risk assessment engine, offering a set of functionalities for the long term prognostic evaluation of diabetes patients.

Main functions are:

- Accept a **risk-assessment request** as an input: the request should contain the patient profile and the diabetes complication(s) of interest
- Process the patient data and compute the risk profile, even when some measurements of the clinical parameters are not provided in the patient profile (**missing information**)
- Return the risk-profile as a function of the personalized risk and probability for developing the complication **over time**

REACTION decided to implement the LTRAM component as a set of independent Web Services (WSs), each one providing an evaluation for a specific diabetes complication. This architecture presents several advantages:

- it is fully **modular**; each WS is independent from the other ones, and new services can be easily added or updated
- WS technology is widely used and accepted nowadays; the development and deployment of applications able to interact with WSs is fast and supported by several automatic tools

- the functionalities of the LTRAM component requires a *stateless* interaction, an operation schema particularly suitable for a Server – Client architecture

The first Web service for assessing the risk of developing Retinopathy has been implemented.

Figure 19 introduces the internal architecture of each single Web Service. Each WS consists of (a) the Web Service Interface, i.e. codes and libraries that implements the required ICT standards (SOAP, WSDL, etc.) for communicating with other application, and (b) the Service Core, that contains the software implementation of the predictive model and the functionalities for dealing with incomplete patient profiles (Missing Information Module).

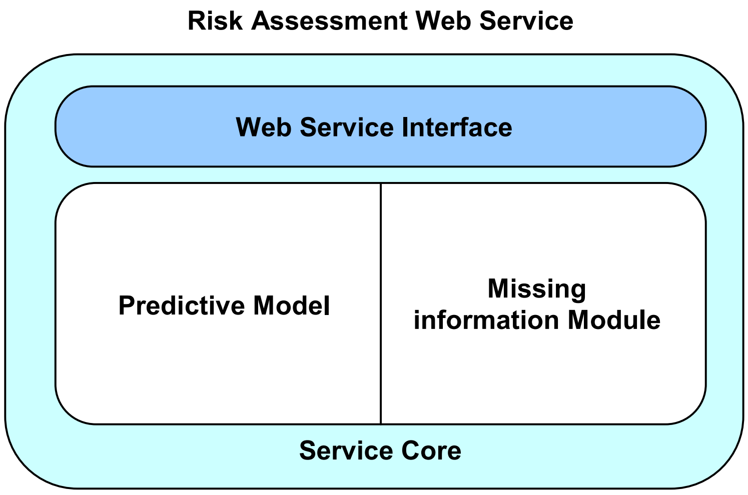


Figure 19: Internal Architecture of a single Web Services handled by the Long-term Risk Manager.

The operation of the Web Service is as follows:

- A predictive model for the time-to-event can be considered as a function $S(t, x_1, \cdots, x_n)$, where $x_1$ , ..., $x_n$, are the values of *n* clinical parameters and *t* is the time. $S(t, x_1, \cdots, x_n)$ is the probability of developing retinopathy *after* time *t* for a patient with the given clinical values for $x_1$ , ..., $x_n$, . Most such models in the literature require that all clinical parameters should be provided for the function to apply. Hence the need for the Missing Information Module. Such models are often implemented as Cox Proportional Hazards models. The model has been induced from the DCCT data by employing advanced and basic feature selection and regression methods for survival analysis.
- To compute the risk for any possible subset of the 80 clinical parameters, one would need to learn a predictive model for each possible such subset (about $10^{24}$ models). The Missing Information Module instead employs a different trick to achieve the same results involving learning a single predictive model, and a Bayesian Network on all clinical variables. In more detail, the module computes the conditional joint probability distribution *p*($x_1$ , ..., $x_n$ | *evidence*) where *evidence* are the clinical values of any subset of the possible input clinical parameters listed in Table 2. The computation of this joint distribution is implemented by doing inference on a Bayesian Network on all clinical parameters. The risk assessment for a patient with clinical parameters in the vector *evidence* is then computed as:

$$S(t, evidence) = \sum_{x_1,\cdots,x_n} S(t, x_1, \cdots, x_n) \cdot p(x_1, \cdots, x_n \mid evidence)$$

Once a request is received, the Missing Information Module computes *p*($x_1$ , ..., $x_n$ | *evidence*) and *S(t, evidence)* by iteratively invoking the predictive model. The Web Service Interface is simply in charge of receiving the request and sending back the evaluation, in XML format.

In order to ensure an easy integration of the LTRAM Component in the REACTION platform, the Web Service Interface is built following the most common WS Standard Protocols: Simple Object Application Protocol (SOAP), Web Service Description Language (WSDL), Universal Description Discovery and Integration (UDDI). The WSDL protocol is employed to describe the functionalities of the Service Provider; the UDDI protocol allows the discovery of the Web Services with the desired functionalities; the SOAP protocol allows the actual interaction among the Web Service (Service Provider) and the Client (Service Requester). The actual implementation of the Web Service Interface is realized by employing the Java–based APACHE AXIS2 technology, and the Web Services are optimized for the APACHE TOMCAT 7 Web Server.

```
- <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
       - <soapenv:Envelope
          xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 - <soapenv:Body>
 - <evaluateRisk xmlns="http://riskEvaluation">
 - <profile>
  <ns1:ID xmlns:ns1="http://patientProfile">Example</ns1:ID>
 - <ns2:attributes xmlns:ns2="http://patientProfile">
          <ns2:name>treatment</ns2:name>
          <ns2:value>1.0</ns2:value>
  </ns2:attributes>
 - <ns3:attributes xmlns:ns3="http://patientProfile">
          <ns3:name>age</ns3:name>
          <ns3:value>25.0</ns3:value>
  </ns3:attributes>
 - <ns4:attributes xmlns:ns4="http://patientProfile">
          <ns4:name>sex</ns4:name>
          <ns4:value>0.0</ns4:value>
  </ns4:attributes>
 - <ns5:attributes xmlns:ns5="http://patientProfile">
          <ns5:name>ret00</ns5:name>
          <ns5:value>-1.0</ns5:value>
  </ns5:attributes>
 - <ns6:attributes xmlns:ns6="http://patientProfile">
          <ns6:name>wpmean</ns6:name>
          <ns6:value>-1.0</ns6:value>
  </ns6:attributes>
 - <ns7:attributes xmlns:ns7="http://patientProfile">
          <ns7:name>retlevel</ns7:name>
          <ns7:value>-1.0</ns7:value>
  </ns7:attributes>
 - <ns8:attributes xmlns:ns8="http://patientProfile">
          <ns8:name>A1c</ns8:name>
          <ns8:value>-1.0</ns8:value>
  </ns8:attributes>
</profile>
</evaluateRisk>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 20: Example of XML request for the Retinopathy Web Service.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
        <evaluateRiskResponse xmlns="http://riskEvaluation">
                <evaluateRiskReturn>
                        <subjectValues>
                                <subjectValues>1.0</subjectValues>
                                <subjectValues>21.0</subjectValues>
                                                [ … ]
                        </subjectValues>
                        <survivalFunction>
                                <survivalFunction>0.95</survivalFunction>
                                <survivalFunction>0.90</survivalFunction>
                                                [ … ]
                        </survivalFunction>
                        <timeOfInterest>
                                <timeOfInterest>2.0</timeOfInterest>
                                <timeOfInterest>2.0</timeOfInterest>
                                                [ … ]
                        </timeOfInterest>
                </evaluateRiskReturn>
        </evaluateRiskResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 21: Example of XML answer from the Retinopathy Web Service.

## 9.2 Short-term Risk Manager

The main goal of the short-term risk management is to assist providing early detection and possibly prevention of hypoglycaemic and hyperglycaemic events. At the same time, proper glycaemic management helps to prevent the development of complications. On the other hand, if the patient has already some complication, another important goal appears, that is the prevention of the aggravation of the complication.

Main functions are to achieve these goals the component highly depends on the Care Plan Component, especially on the risk management plan which defines the necessary monitoring strategy, including home monitoring, periodic laboratory tests and examinations and quantitative data collection by questionnaires. The STRM Component monitors if the patient follows the care plan. Moreover it continuously analyses the collected data in order to detect problems, e.g. harmful patterns in the blood glucose levels or short-term risks related to the complications.

Input:

- Self-monitoring results
- Care plan
- Treatment (insulin doses injected or number of pills taken)
- lifestyle (diet, exercise, stress)
- complication-specific parameters

Knowledge:

This component requires short-term risk models for blood glucose control and various complications.

Output:

- identified blood glucose trends and patterns
- short-term risk predictions concerning the complications

Figure 22 provides an overview on the internal components of the Short-term risk management.

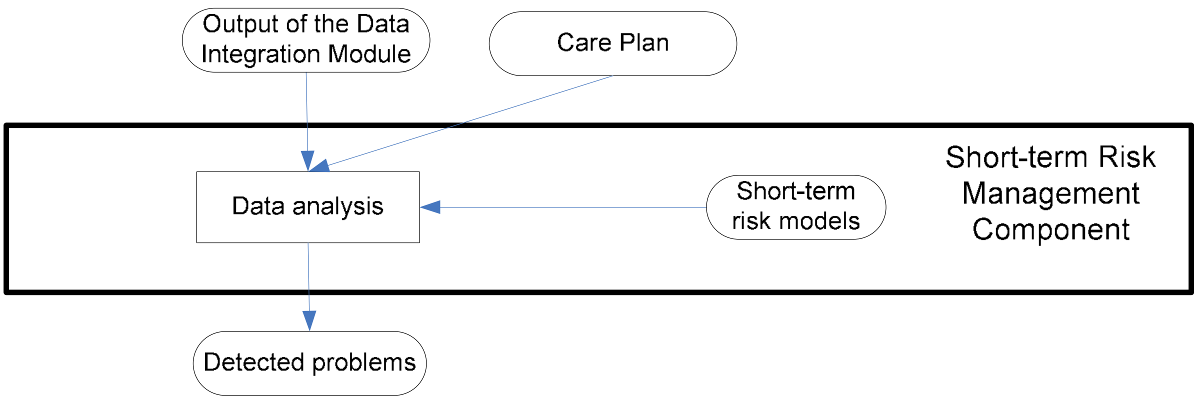

Output of the Data Integration Module
Care Plan
Data analysis
Short-term risk models
Short-term Risk Management Component
Detected problems

Figure 22 Short-term Risk Management Component

The following subsections describe how the responsibilities are distributed between the internal components.

| | |
|---|---|
| *Hypoglycaemic state* | *Number of hypoglycaemic readings and/ or episodes is above a threshold* in a week* |
| *Hyperglycaemic state* | *Number of hyperglycaemic readings is above a threshold in a week* |
| *Oscillating state* | *Number of both hypo- and hyperglycaemic readings are above a threshold* |
| *"Somogyi" effect* | *High reading within 12 hours after a hypo* |
| *Dawn phenomenon* | *Reading before breakfast is high and no hypo was detected at night* |
| *Too high post-meal increase* | *The difference between a pre- and post-meal reading is above a threshold* |
| *Too much increase or decrease between two meals* | *The difference of two pre-meal readings is above a threshold (positive or negative)* |
| *Hypo- or hyperglycaemia at a time of day* | *The mean of the readings is above or below a threshold at a time of day (at least three days interval)* |
| *Not enough data* | *The patient does not provide measurements according to the monitoring plan* |

## 9.3 Risk Classification Manager

The goal of the component is to assign the patients into one of the risk groups. Four risk groups are proposed: low, moderate, elevated and high risk group.

The main principles for the classification are the following (for some cases the development of the exact classification criteria needs further study):

- Patients who already have complications are assigned to one of the two high risk groups. Those who are currently well controlled and coping well could be assigned to the 3rd (elevated) group while those with poorly controlled diabetes and/or poor coping belong to the highest risk group
- Patients without complications are classified based on their
  - Blood glucose control (last HbA1c and self-monitoring results)
  - Number of risk factors
  - Psychological-mental status (coping, knowledge level)

| Risk group | **Low risk**<br>BG well controlled<br>Max. 2 risk factors | **Moderate risk**<br>BG fairly well controlled<br>More than 2 risk factors | | **Elevated risk**<br>BG poorly controlled<br>No complications<br>Bad coping | **High risk**<br>BG poorly controlled<br>Complications<br>and/or bad coping |
|---|---|---|---|---|---|
| | | Good coping | Bad coping | | |

Input:

- existing complications
- blood glucose control (HbA1c, self-monitoring trends, patterns)
- risk factors
- coping level

The inputs may be provided by the Data Integration Module.

Knowledge:

This component requires a classification model which describes the rules for classification.

Output:

- Risk class: low/medium/elevated/high

Figure 10 provides an overview on the internal components of the Risk Classification Component.

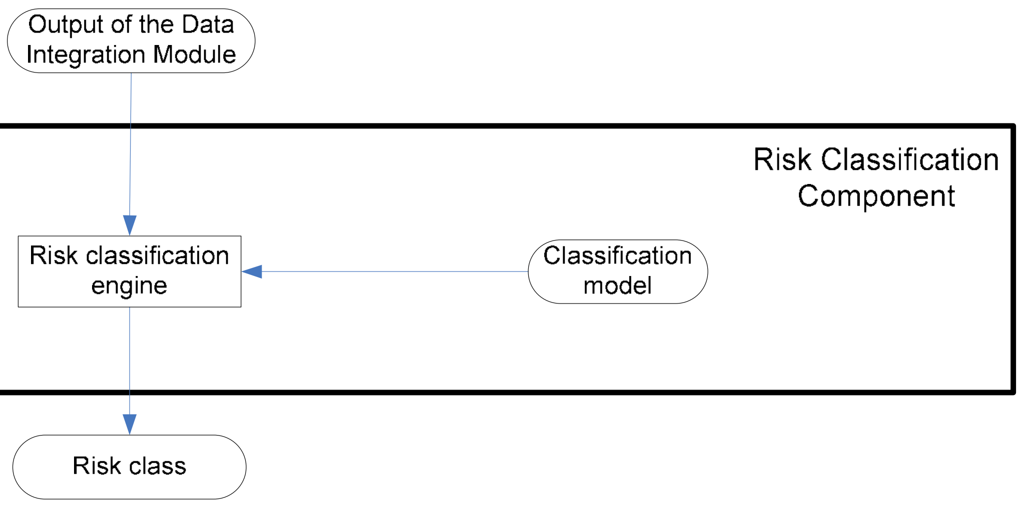
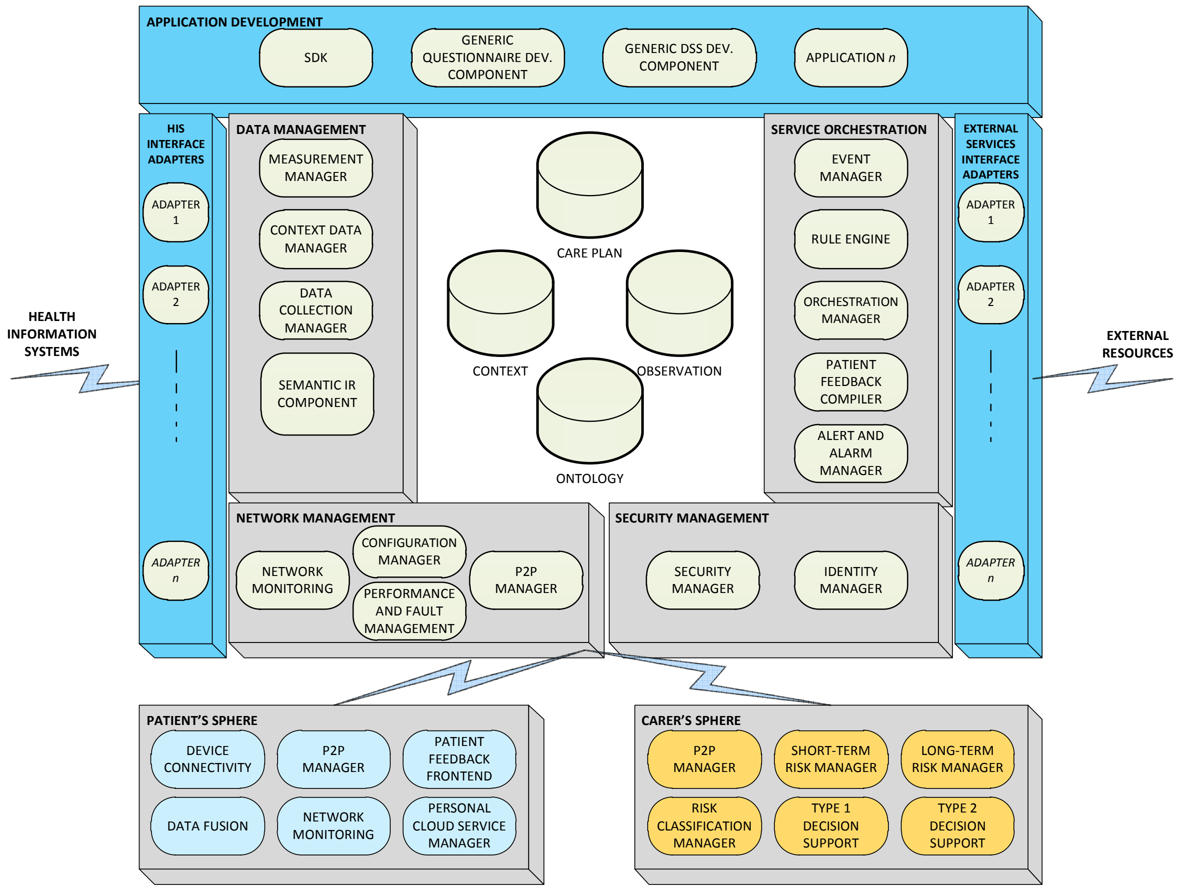

Output of the Data Integration Module
Risk Classification Component
Risk classification engine
Classification model
Risk class

Figure 23 Risk Classification Component

The following subsections describe how the responsibilities are distributed between the internal components. This component uses the outputs of the Data Integration Module and the Care Plan. The outputs can be used by the decision support modules for providing advices.

# 10 Application Development and Adaptation



## 10.1 Generic Decision Support Development Component

Decision support can provide several valuable services to the users, e.g. support the physician in the selection of the most appropriate treatment or give advices concerning the management of the patient, e.g. suggests modifications to the care plan. These services require the use of knowledge, which can be represented in a variety of formats.

This generic component supports the development of various decision support modules.

The development of such a decision support (DS) module requires the following elements:

- An interface which supports the specification of the DS module to be developed, including:
    - the selection of the knowledge representation type to be used (e.g. rules, decision tree, guideline) and the definition of the knowledge necessary for the decision making
    - the definition of the necessary input parameters (data items), e.g. it can use the outputs of the Data Integration Module
    - the definition of the expected outputs – it can be e.g. advice, diagnosis, prediction
    - the selection of the user interface to be used by the DS module
- Data structures for the above elements
- A generator for the construction of the DS module from the above inputs

The output of the component is a particular decision support module which can be used by the applications. For example a module may be developed for the analysis of the data collected by questionnaires and generating conclusions.

Figure 24 provides an overview on the internal components.

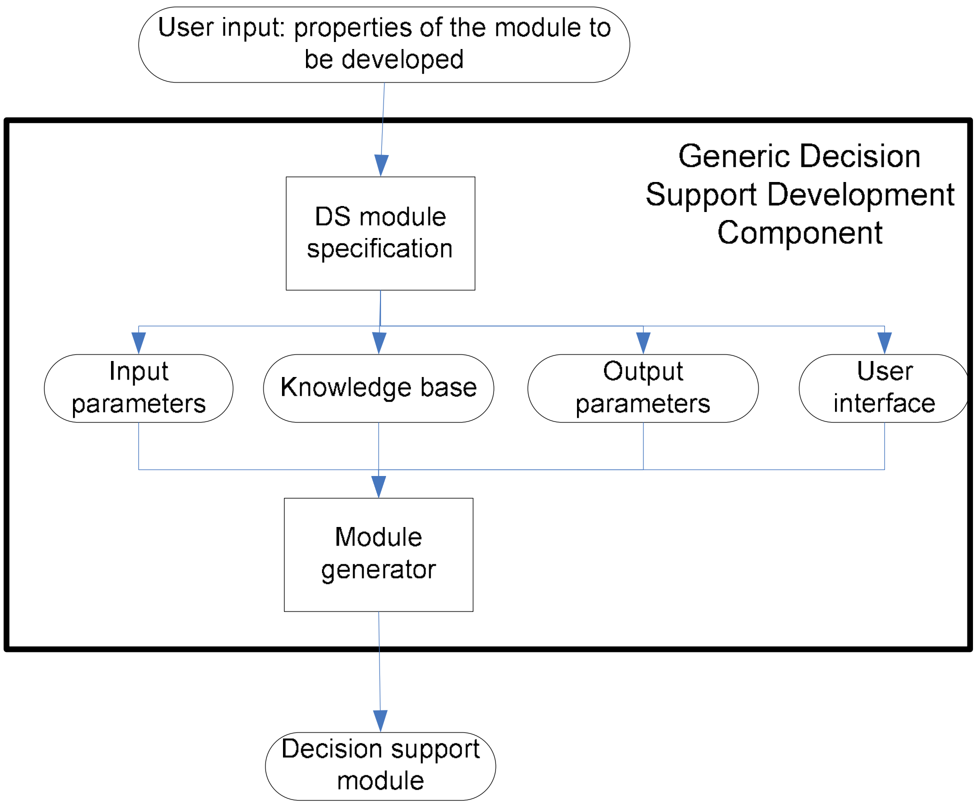


Figure 24 Generic Questionnaire development Component

The following subsections describe how the responsibilities are distributed between the internal components.

## 10.2 Generic Questionnaire Development Component

This component provides a generic framework for the development of various questionnaires. It will be used for developing appropriate questionnaires e.g. to collect data about the patients' diet and physical activity.

The component's main elements are the following:

- Interface for the definition of the type of the questionnaire to be developed. It allows the definition of questions and their relationships, potential answers and their type.
- Interface for the definition of the text generation. It allows the definition of the text generation rules and the structure of the report. (E.g. it permits to define the classification of the questions by their importance).
- Questionnaire database for the storage of the above defined questionnaire elements.
- Text generation database for the storage of the above defined components
- Generic user interface to support the questionnaire development process (collection of the necessary data, generation of the structure of the questionnaire, activation of the questionnaire interface and its service programs, actualisation/fulfilment of the questionnaire, testing).
- Generic user interface to support the report generator development process (collection of the necessary data, development of the report generator structure, activation of the report

generator interface and its service programs, actualisation/fulfilment of the report generator, testing).

The output of the component is the questionnaire which can be used by the applications for data collection.

Figure 25 provides an overview on the internal components.

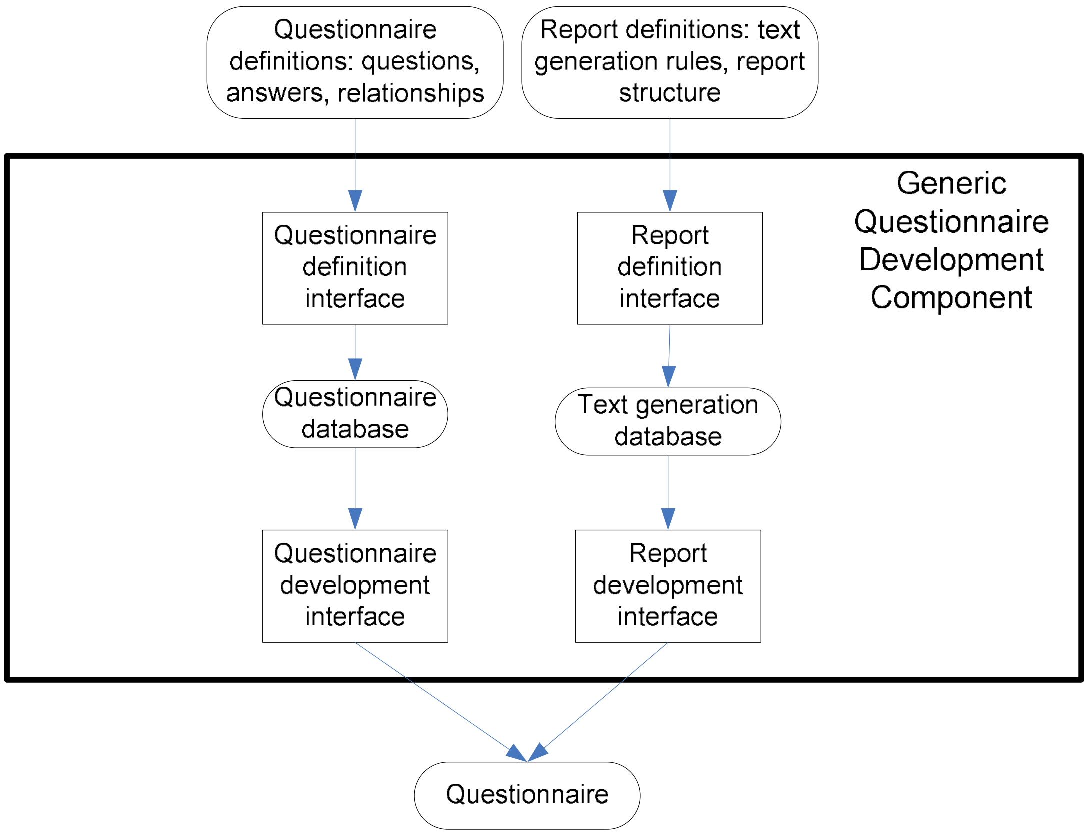

Questionnaire definitions: questions, answers, relationships
Report definitions: text generation rules, report structure
Generic Questionnaire Development Component
Questionnaire definition interface
Report definition interface
Questionnaire database
Text generation database
Questionnaire development interface
Report development interface
Questionnaire

Figure 25 Generic Questionnaire development Component

The following subsections describe how the responsibilities are distributed between the internal components.

The data collected by the questionnaires will be managed by a special Qualitative Data Management Module (QDMM), which collects and integrates the reports of the selected questionnaires.

A special Data Integration Module (DIM) will be responsible for the integration of the summary report of a questionnaire or the integrated summary report of a set of questionnaires, the specified data from the EPR and the devices. This module provides a semantically consistent set of mixed (qualitative and quantitative) data. The DIM may provide the input for the Decision Support Modules and the Risk Classification and Short-term Risk Management Components.

## 10.3 Interface Adapters

The purpose of all Interface Adapters is to specifically assist in the device specific services that can be integrated with external services, such as knowledge extraction, accessing an EPR or providing feedback to a carer, merged with workflow and resource scheduling services and supplied with security model and authentication services.

Further, it will make health information from outside the REACTION platform accessible. Comprehensive physiological models can be access via web interfaces and the result fed back to the physician responsible for the case. General medical and clinical information in the open internet can be semantically queried and the results put together in a comprehensive report on risk assessment, which will be personalised and presented to the patient in the self-management scheme.

This will support the new models of business constellations that will be explored in the REACTION project. These include private public partnerships, collaboration pharmaceutical companies as innovation drivers and bringing together payers, providers and patients in new constellations. Interface Adapters will show how to share proprietary information across organisational barriers, involve and transform the patient from a passive health information provider to an active information user, and safe handling of the massive flow of information and intellectual property rights to healthcare information.

Main functions are:

- Providing interfaces to access internal existing information systems
- Providing interfaces to access and communicate with external systems and services, such as cloud-based services.

## 11 Summary

In this deliverable we have summarised the components and tools that constitutes the REACTION Software Developer Kit. It includes all the different components and managers that interplay in order to fulfil the orchestration of services as means to support cost-effective development of a broad range of innovative healthcare applications. The REACTION SDK demonstrate how applications can be built on top of the REACTION platform’s integrated approach to improved long term management of diabetes through continuous blood glucose monitoring, monitoring of significant events, monitoring and predicting risks and/or related disease indicators, decision on therapy and treatments, education on life style factors such as obesity and exercise and, ultimately, automated closed-loop delivery of insulin.

The REACTION Software Development Kit (SDK) allows developers to rapidly create new networked applications on the REACTION platform while providing solution developers with a high-level interface for innovative monitoring applications with embedded intelligence and closed loop feedback provisioning using the REACTION platform.

# Appendix

Table 1 Measurement Manager

| | | |
|---|---|---|
| Main functions: | • Receive measurements and update database.<br>• Retrieve different vital signs measurements (e.g. blood Glucose, blood pressure, weight scale $SpO_2$, etc.) for patient Retrieve different vital signs measurements for specified time period.<br>• Provide conversions of measurements to different formats (e.g. ORU to XML and back).<br>• Delete measurements.<br>• Checks measurement so it complies with patient informed consent.<br>• Acknowledge the received and successful storage of data. | |
| Processed requirements: | REACTION-70 | Processing of multi-parametric clinical and non-clinical data from different sources. |
| | REACTION-265 | The clinical parameters to be measured must be specified. |
| | REACTION-338 | All data entered must be checked for format, consistency and validity. |
| Components | HL7Parser, Database Manager. | |
| Dependencies | P2P Manager, Event Manager, Observation Database, Data Collection Manager | |
| Interface | n/a | |

Table 2 Context Data Manager

| | | |
|---|---|---|
| Main functions: | • Retrieve context associated with a patient and a set of measurements.<br>• Receive and store context associated with a patient and environmental data.<br>• Update context and delete context associated with a patient. | |
| Processed requirements: | REACTION-82 | Contextualized and personalized feedback to patients and carers. |
| | REACTION-237 | Annotation of blood glucose values, especially in inpatient environment. |
| | REACTION-371 | Use of activity patterns for context annotations. |
| | REACTION-372 | Context of observations. |
| Components | Database Manager, Context Analyser | |
| Dependencies | P2P Manager. Event Manager, SIR, Data Collection Manager, Context Database | |
| Interface | n/a | |

Table 3 Data Collection Manager

| | |
|---|---|
| Main functions: | • Exchanges basic configuration information with client gateways, e.g. what devices should be there |

| | | |
|---|---|---|
| | • Adjusts transfer frequency after its performance quality<br>• Provides transparent communication to and from clients and other components<br>• Receive measurement<br>• Receive context data<br>• Receive device events and info<br>• Receive patient info<br>• Relate measurement to patient<br>• Relate context to patient<br>• Publish abnormal device state (events)<br>• Relate incoming patient info to stored patient<br>• Keep incoming message queue(s)<br>• Provide audit trail by logging incoming messages<br>• Unpacks data fused collections and updates context and measurement databases accordingly<br>• Provides client gateway statistics, e.g. number of messages processed per day.<br>• Checks with client gateways which devices are active and their QoS. | |
| Processed requirements: | REACTION-17 | Configurable data transfer frequency. |
| | REACTION-141 | The user should have choices regarding all data collection activities concerning his personal data. |
| | REACTION-236 | Blood glucose measurements in Inpatient environment. |
| | REACTION-244 | The data management and the user interface shall allow the insertion of specific interfering drugs (including their dosage). The dosage of insulin shall vary with these drugs. |
| | REACTION-347 | Continuous blood glucose monitoring. |
| | REACTION-356 | Manual data insertion. |
| | REACTION-410 | Collecting measured data ("many to one" traffic pattern). |
| Components | Data Receiver, Data Processor, Message Router, Message Queue, Log Manager, Event Publisher. | |
| Dependencies | Event Manager, Identity Manager, Network and Fault Manager. | |
| Interface | n/a | |

Table 4 Semantic IR Component

| | | |
|---|---|---|
| Main functions: | • Data base of EPRs – since EPR usually contains data represented in the form of natural language text,<br>• The Cochrane archive,<br>• Archives of guidelines | |
| Processed requirements: | REACTION-74 | Formalization of pre-existing clinical data (semantic structure). |
| | REACTION-346 | Knowledge Discovery from unstructured clinical text information. |
| | REACTION-381 | Definition of a common ontology to refer to data, metadata, interfaces and models. |

| | | |
|---|---|---|
| | REACTION-459 | Ontologies and data management designed for the storage and multi-user availability of all relevant information, actions, treatments, events. |
| | REACTION-463 | Context management for clinical (lab) values. |
| | REACTION-467 | Semantics based data management |
| Components | GUI, Parser, Semantic Lexicon, Generator of the Meaning Representation, Pre-search Engine, Search Engine. | |
| Dependencies | The component requires access to the repositories where it will perform searches. | |
| Interface | n/a | |

Table 5 Event Manager

| | | |
|---|---|---|
| Main functions: | • Subscription support allowing clients to subscribe to published events via a topic-based publish/subscribe scheme<br>• Publication support allowing client to publish event on topics<br>• Routing events to subscribed clients<br>• Event Core manages persistent subscriptions, publication to subscription matching etc.<br>• Interfacing to Network Manager (e.g., broadcast-, multicast-, or gossiping-based dissemination)<br>• Storing events<br>• Priorities events<br>• Retry sending events. | |
| | REACTION-24 | Logging of events from components. |
| | REACTION-444 | 6-month clinical checks. |
| Components | GUI, Parser, Semantic Lexicon, Generator of the Meaning Representation, Pre-search Engine, Search Engine. | |
| Dependencies | P2P Manager, Subscription Manager, Notification Manager, Publication Manager, Event Core. | |
| Interface | *string getSubscriptions()*<br>*bool clearSubscriptions(string subscriber)*<br>*bool setPriority(string topic, int priority)*<br>*bool subscribe(string topic, string endpoint, int priority)*<br>*bool subscribeWithHID(string topic, string hid, int priority)*<br>*bool unsubscribe(string topic, string subscriber)*<br>*Event[] failedNotifies(string topic, string endpoint, bool clearFailes)*<br>*Event[] failedNotifiesWithHID(string topic, string hid, bool clearFailes)* | |

Table 6 Rule Engine

| | |
|---|---|
| Main functions: | • Add rule<br>• Update rule<br>• Delete rule<br>• Evaluate rules<br>• Get rule report |

| | | |
|---|---|---|
| | • Get patient rules<br>• Associate alarm and alerts with patient rules | |
| Processed requirements: | REACTION-179 | Daily data review by clinicians or Telehealth support team. |
| | REACTION-374 | Annual clinical checks. |
| | REACTION-419 | Set of event rules. |
| | REACTION-425 | Set of action rules. |
| Components | IoTDevice | |
| Dependencies | Orchestration Manager, Identity Manager, Event Manager. | |
| Interface | `void EvaluateRules(System.String IoTEvent);`<br>`System.String GetRuleSet();`<br>`void ListenToEvents(System.String IoTEvent);`<br>`void SetRuleSet(System.String ruleset);`<br>`void StopListenToEvents(System.String IoTEvent);` | |

Table 7 Orchestration Manager

| | | |
|---|---|---|
| Main functions: | • Add orchestration scheme<br>• Update orchestration scheme<br>• Delete orchestration scheme<br>• Start orchestration<br>• Execute orchestration actions<br>• Stop orchestration<br>• Log and notify about orchestration events and actions<br>• Notify completion of orchestration<br>• Generate orchestration report | |
| Processed requirements: | REACTION-23 | Clinician generated feedback to patient. |
| | REACTION-202 | Setup remote patient monitoring scheme. |
| | REACTION-231 | End of process for the diabetic patient in the inpatient environment. |
| | REACTION-258 | Automated transfer of patient related data from the hospital information system. |
| | REACTION-403 | Each entity in the Reaction platform MUST be representable by a digital identity. |
| | REACTION-404 | Workflow Orchestration Manager. |
| | REACTION-441 | Basic workflow in Inpatient environment. |
| | REACTION-468 | Provide regular update of data model for Health Care profile. |
| Components | Schedule manager, Workflow Execution Manager, Application Service Manager, Application Device Manager | |
| Dependencies | P2P Manager. Event Manager, SIR, Data Collection Manager, Context Database, Event Manager, P2P Manager, Identity Manager, Rule Engine | |
| Interface | n/a | |

Table 8 Alert and Alarm Handler

| | | |
|---|---|---|
| Main functions: | • Add orchestration scheme<br>• Update orchestration scheme<br>• Delete orchestration scheme<br>• Start orchestration<br>• Execute orchestration actions<br>• Stop orchestration<br>• Log and notify about orchestration events and actions<br>• Notify completion of orchestration<br>• Generate orchestration report | |
| Processed requirements: | REACTION-160 | Alerts for the annual and 6-month clinical checks. |
| | REACTION-161 | Alarm system- reminder to perform measurements. |
| | REACTION-193 | Alarm & alert generation. |
| | REACTION-217 | Acquired values in the alarm range. |
| | REACTION-380 | Set of alerts and reminders. |
| | REACTION-448 | Alert / notification messages should be short enough in order to be delivered as SMS messages if necessary. |
| Components | n/a | |
| Dependencies | Event Manager, Rule Manager. | |
| Interface | n/a | |

Table 9 P2P Manager

| | |
|---|---|
| Main functions: | • Creates and overlay P2P network, where all the IoT-enabled devices appear directly interconnected, no matter if they are behind a NAT (Network Address Translator) or Firewall.<br>• Provides indirection architecture for addressing Web Services hosted by IoT devices using the HID addressing mechanism. Each service is identified in IoT through an HID, which is a global and unique identifier. The P2P Manager provides interfaces for other managers, applications and IoT devices for HID creation, modification and deletion. It also offers the possibility to select the transport protocol for the service invocation between TCP, UDP and Bluetooth.<br>• Provides a transport mechanism over the overlay P2P network for invoking Web Services hosted by IoT devices (SOAP Tunnelling) using the HID addressing mechanism. The SOAP messages addressed to an HID are routed by the P2P Manager through the overlay network to the P2P Manager hosting the service. Therefore, using the SOAP Tunnelling and the P2P Manager any device or application is able to transparently publish and consume services anywhere, anytime, breaking the network interconnectivity barriers and independently of the service endpoint location.<br>• Provides a transport mechanism over the overlay P2P network for multimedia content exchange between UPnP AV or DLNA devices.<br>• Provides session management mechanisms between HIDs during service invocations.<br>• Provides time reference synchronization between different P2P Managers. |

| | | |
|---|---|---|
| | • Provides a status page for developers, which the developer can use for monitoring dynamic information about the Internet-of-Things Network and the HIDs available. | |
| Processed requirements: | REACTION-1 | Internet communication between patient home and primary/secondary healthcare structures based on public wired or wireless network. |
| | REACTION-28 | Network interoperability. |
| | REACTION-172 | Automatic transmission of glucose values from POCT system to REACTION platform (time-critical!). |
| | REACTION-173 | Platform should allow ubiquitous access to end-users and sharing of information among caregivers (multiuser access to relevant data). |
| | REACTION-354 | Data/messages exchanged between the Reaction Host Client and the Reaction Device Hosting Server MUST be authentic (message authentication), with integrity, and confidential. |
| | REACTION-414 | Communication between the Reaction Hosting Client and the Reaction Device Hosting Server MUST be authentic (entity authentication), with integrity, and confidential. |
| | REACTION-451 | In-hospital prototype communication with REACTION platform. |
| | REACTION-453 | Communication interface between REACTION Client and REACTION Server. |
| Components | Routing Manager, Session Manager, HID Manager, Time Manager, Backbone Manager, SOAP Tunnelling, SecurityLibrary | |
| Dependencies | P2P Manager. Event Manager, SIR, Data Collection Manager, Context Database, Event Manager, P2P Manager, Identity Manager, Rule Engine | |
| Interface | eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.[static initializer] () `[static, package]`<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.init ()<br>Instantiates the **network** manager submanagers and registers the servlets.<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.activate (ComponentContext *context*) `[protected]`<br>Activate method.<br>Parameters:<br>*context* the bundle's execution context<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.deactivate (ComponentContext *context*) `[protected]`<br>Deactivate method.<br>Parameters:<br>*context* the bundle's context<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.bindTrustManager (TrustManager *trustManagerService*) `[protected]`<br>Binds the Trust Manager.<br>Parameters:<br>*trustManagerService* the Trust Manager to bind.<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.unbindTrustManager (TrustManager *trustManagerService*) `[protected]`<br>Unbinds the Trust Manager.<br>Parameters:<br>*trustManagerService* the Trust Manager to unbind<br>void | |

| | |
|---|---|
| | eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.bindInsideSecurity (InsideHydra *insideHydraSecurity*) `[protected]`<br>Binds InsideHydraSecurity.<br>Parameters:<br>*insideHydraSecurity* to use<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.unbindInsideSecurity (InsideHydra *insideHydraSecurity*) `[protected]`<br>Unbinds InsideHydraSecurity.<br>Parameters:<br>*insideHydraSecurity* to use<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.cryptoBind (CryptoManager *cryptoManagerService*) `[protected]`<br>Binds the Crypto Manager.<br>Parameters:<br>*cryptoManagerService* the Crypto Manager to bind<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.cryptoUnbind (CryptoManager *cryptoManagerService*) `[protected]`<br>Unbinds the Crypto Manager.<br>Parameters:<br>*cryptoManagerService* the Crypto Manager to unbind<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.addTransport (ServiceReference *transportRef*) `[protected]`<br>Adds a transport.<br>Parameters:<br>*transportRef* the transport to add<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.deleteTransport (ServiceReference *transportRef*) `[protected]`<br>Deletes a transport.<br>Parameters:<br>*transportRef* the transport to delete<br>HashMap eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getTransports ()<br>Gets the list of transports.<br>Returns:<br>the list of transports<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.bindRemoteWSClientProvider (RemoteWSClientProvider *wsclientProvider*) `[protected]`<br>Binds a remote WS Client Provider.<br>Parameters:<br>*wsclientProvider* the remote WS client provider to bind<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.unbindRemoteWSClientProvider (ServiceReference *transportRef*) `[protected]`<br>Unbinds a remote WS Client Provider.<br>Parameters:<br>*transportRef* the transport<br>Dictionary eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getConfiguration () |

| | |
|---|---|
| | Gets the configuration.<br>Returns:<br>the configuration<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.setConfiguration (String *key*, String *value*)<br>Sets the configuration.<br><u>Parameters</u>:<br>*key* key value<br>*value* value value<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.configurationBind (ConfigurationAdmin *cm*) `[protected]`<br>Binds a configuration.<br><u>Parameters</u>:<br>*cm* the configuration admin<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.configurationUnbind (ConfigurationAdmin *cm*) `[protected]`<br>Unbinds a configuration.<br><u>Parameters</u>:<br>*cm* the configuration admin<br>ComponentContext eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getContext ()<br>Gets the context.<br>Returns:<br>the context<br>CryptoManager eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getCryptoManager ()<br>Gets the Crypto Manager.<br>Returns:<br>the Crypto Manager<br>InsideHydra eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getSecurityLibrary ()<br>HIDManagerApplication eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHIDManagerApplication ()<br>Gets the HID Manager Application.<br>Returns:<br>the HID Manager Application<br>PipeSyncHandler eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getPipeSyncHandler ()<br>Gets the pipe synchronization handler.<br>Returns:<br>the pipe synchronization handler<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.stop ()<br>Stop.<br>String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.startNM ()<br>Returns:<br>"OK"<br>Deprecated:<br>String |

eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.stopNM ()

Returns:

""

Deprecated:

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.openSession (String *senderHID*, String *receiverHID*)

It allows opening a session between two HIDs for data exchange. A session will be established between the senderHID and the receiverHID. The sessions by default expires 60000 milliseconds.

Parameters:

| | |
|---|---|
| *senderHID* | The HID of the sender |
| *receiverHID* | The HID of the receiver |

Returns:

The session identifier (uuid) for the generated session

NMResponse eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.sendData (String *sessionID*, String *senderHID*, String *receiverHID*, String *data*) throws RemoteException

Sends data.

Parameters:

| | |
|---|---|
| *sessionID* | the session id |
| *senderHID* | the sender HID |
| *receiverHID* | the receiver HID |
| *data* | the data to send |

Returns:

the NM response

Deprecated:

Use SOAP tunneling instead

NMResponse eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.receiveData (String *sessionID*, String *senderHID*, String *receiverHID*, String *data*) throws RemoteException

Receive data.

Parameters:

| | |
|---|---|
| *sessionID* | the session id |
| *senderHID* | the sender HID |
| *receiverHID* | the receiver HID |
| *data* | the data |

Returns:

the NM response

Deprecated:

Use SOAP tunneling instead

void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.closeSession (String *sessionID*)

It allows to close a session using a session identifier. If the session is not closed, it will be closed after the expiration time (by default 60000 millisecond)

Parameters:

| | |
|---|---|
| *sessionID* | The session identifier. |

java.lang.String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getSessionParameter (String *sessionID*, String *key*)

This method allows to get stored data in the session object.

Parameters:

| | |
|---|---|
| *sessionID* | The sessionID of the session where the data is stored |
| *key* | The key for the requested parameter |

Returns:

The value of the requested parameter or null if the sessionID or the parameter doesn't exist.

void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.setSessionParameter (String *sessionID*, String *key*, String *value*)

This method allows to store data (key-value pair) in a session object.

Parameters:

| | |
|---|---|
| *sessionID* | The sessionID of the session where the data will be stored. |
| *key* | The key on which the data will be stored. |
| *value* | The data to be stored |

java.util.Vector eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.synchronizeSessionsList (String *senderHID*, String *receiverHID*)

Operation to synchronize the clientSessionsList with the serverSessionsList of this Network Manager.

Parameters:

| | |
|---|---|
| *senderHID* | The HID of the client that need to synchronize its clientSessionsList with the serverSessionsList of this Network Manager |
| *receiverHID* | The HID of the server HID |

Returns:

A vector that contains the sessionID to be deleted

void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.addSessionRemoteClient (String *sessionID*, String *senderHID*, String *receiverHID*) throws RemoteException

Parameters:

| | |
|---|---|
| *sessionID* | the session ID |
| *senderHID* | the sender HID |
| *receiverHID* | the receiver HID |

Deprecated:

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.createHID (long *contextID*, int *level*)

Operation to create an HID with a predefined contextID and level It calls the HID Manager for creating the HID and it will be added to the idTable

Parameters:

| | |
|---|---|
| *contextID* | The desired context ID to be created |
| *level* | The desired level |

Returns:

The **String** representation of the HID

Deprecated:

This method will be deleted in the next release

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.createHID ()

Operation to create an HID without any context It calls the Identity Manager for creating the HID and it will be added to the idTable

Returns:

The **String** representation of the HID

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.createHIDwDesc (long *contextID*, int *level*, String *description*, String *endpoint*) throws RemoteException

Operation to create an HID with a predefined contexID and level and providing a description for the HID (searching purposes) and the endpoint of the service behind it (for service invocation)

Parameters:

| | |
|---|---|
| *contextID* | The desired context ID to be created |
| *level* | The desired level |
| *description* | The description associated with this HID |
| *endpoint* | The endpoint of the service (if there is a service behind) |

Returns:

The **String** representation of the HID

Deprecated:

It is better to use from now **createCryptoHID(String xmlAttributes, String endpoint)**

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.createHIDwDesc (String *description*, String *endpoint*) throws RemoteException

Operation to create an HID providing a description for the HID (searching purposes) and the endpoint of the service behind it (for service invocation).

Parameters:

| | |
|---|---|
| *description* | The description associated with this HID |
| *endpoint* | The endpoint of the service (if there is a service behind) |

Returns:

The **String** representation of the HID.

Deprecated:

It is better to use from now **createCryptoHID()**.

CryptoHIDResult eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.createCryptoHID (String *xmlAttributes*, String *endpoint*)

Operation to create a crypto HID providing the persistent attributes for this HID and the endpoint of the service behind it (for service invocation). The crypto HID is the enhanced version of HIDs, that allow to store persistent information on them (through certificates) and doesn't propagate the information stored on it. In order to exchange the stored information, the Session Domain Protocol is used. It returns a certificate reference that point to the certificate generated. The next time the HID needs to be created, using the same attributes, the certificate reference can be used.

Parameters:

*xmlAttributes* The attributes (persistent) associated with this HID. This attributes are stored inside the certificate and follow the Java **java.util.Properties** xml schema.

*endpoint* The endpoint of the service (if there is a service behind).

Returns:

A **eu.linksmart.network.ws.CrypyoHIDResult** containing **String** representation of the HID and the certificate reference (UUID)

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.createCryptoHIDfromReference (String *certRef*, String *endpoint*)

Operation to create an crypto HID providing a certificate reference (from a previously created cryptoHID) and an endpoint The crypto HID is the enhanced version of HIDs, that allow to store persistent information on them (through certificates)

Parameters:

| | |
|---|---|
| *certRef* | The certificate reference from a previously generated cryptoHID. |
| *endpoint* | The endpoint of the service (if there is a service behind). |

Returns:

The **String** representation of the HID.

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.renewHID (long *contextID*, int *level*, String *hid*)

Renews an HID

Parameters:

| | |
|---|---|
| *contextID* | the context ID |
| *level* | the level |
| *hid* | the HID to renew |

Returns:

the HID renewed

Deprecated:

boolean eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.renewHIDAttributes (String *ownerUUID*, String *hid*, String *newXMLAttributes*) throws RemoteException

Operation to modify the attributes associated with an HID. This method provides the means for HID owners to modify the attributes associated with an already generated HID. In order to avoid security threats, the requester has to provide the unique ownerID that is associated with this HID. This ownerID is given to the HID creator in the HID creation response (still to be implemented).

Parameters:

| | |
|---|---|
| *ownerUUID* | The owner UUID associated with the HID. For now, use just the HID |
| *hid* | The HID to be modified |
| *newXMLAttributes* | The new attributes following the properties format. See createCryptoHID |

for more information about the format of this attributes.

Returns:

The result of the operation in **boolean** format. Returns false if the HID could not be found or the ownerID is not valid.

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.renewHIDInfo (String *description*, String *endpoint*, String *hid*) throws RemoteException

Operation to renew the information associated with an HID (description or endpoint). It can be used for example, to change the transport protocol for service invocation.

Parameters:

| | |
|---|---|
| *description* | The new descritpion (or null if no change is required) |
| *endpoint* | The new endpoint of the service (or null if no change is required) |
| *hid* | The hid on which the change is requested |

Returns:

The **String** representation of the HID.

boolean eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.renewHIDEndpoint (String *ownerUUID*, String *hid*, String *endpoint*) throws RemoteException

Operation to change the endpoint associated with this HID. This allows protocol switching or any application that wants to modify dynamically the endpoint on which the service invocations will be forwarded

Parameters:

| | |
|---|---|
| *ownerUUID* | The owner UUID associated with the HID. For now, use just the HID |
| *hid* | The HID on which the endpoint needs to be modified |
| *endpoint* | The new endpoint for this HID |

Returns:

The result of the operation in **boolean** format. Returns false if the HID could not be found or the ownerID is not valid.

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.addContext (long *contextID*, String *hid*)

Adds a context

Parameters:

| | |
|---|---|
| *contextID* | the context HID |

| | |
|---|---|
| *hid* | the HID |

Returns:

the context.

Deprecated:

java.util.Vector eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHIDs ()

Operation to retrieve all HIDs in the LinkSmart P2P **network**

Returns:

A **java.util.Vector** containing all the HIDs in the LinkSmart Network

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHIDsAsString () throws RemoteException

Operation to retrieve all HIDs in the LinkSmart P2P **network** in String format

Returns:

A **String** containing all HIDs in the LinkSmart P2P **network**, separated by commas.

java.util.Vector eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHostHIDs ()

Operation to retrieve the HIDs associated with this Network Manager (local HIDs)

Returns:

A **java.util.Vector** containing all the local HIDs of this Network Manager.

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHostHIDsAsString () throws RemoteException

Operation to retrieve the HIDs associated with this Network Manager (local HIDs) in String format

Returns:

A **String** containing all the local HIDs of this Network Manager, separated by commas.

java.util.Vector eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getContextHIDs (String *contextID*, String *level*)

Get the context HIDs

Parameters:

| | |
|---|---|
| *contextID* | the context ID |
| *level* | the level |

Returns:

the list of context HIDs

Deprecated:

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getContextHIDsAsString (String *contextID*, String *level*) throws RemoteException

Get the context HIDs in string format

Parameters:

| | |
|---|---|
| *contextID* | the context ID |
| *level* | the level |

Returns:

the list of context HIDs in string format

Deprecated:

Vector eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHIDsbyDescription (String *description*) throws RemoteException

Operation to retrieve all HIDs in the LinkSmart P2P **network** that match a description. The description allows inexact matches using magic characters

For example:

getHIDsbyDescription("Network*") -> Will return all the HIDs with description starting with Network getHIDsbyDescription(*Peter'sPortable*") -> Will return all the HIDs with description containing Peter'sPortable

Parameters:

*description* The description to match against

Returns:

A **java.util.Vector** containing all the HIDs in the LinkSmart Network that match the given description

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHIDsbyDescriptionAsString (String *description*) throws RemoteException

Operation to retrieve all HIDs in the LinkSmart P2P **network** (in String format) that match a description. The description allows inexact matches using magic characters

For example:

getHIDsbyDescription("Network*") -> Will return all the HIDs with description starting with Network getHIDsbyDescription(*Peter'sPortable*") -> Will return all the HIDs with description containing Peter'sPortable

Parameters:

*description* The description to match against

Returns:

A **String** containing all the HIDs in the LinkSmart P2P **network** that match the description

String [] eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHostHIDsbyDescription (String *description*) throws RemoteException

Operation to retrieve all HIDs in the LinkSmart P2P **network** (in String format) that match a description. The description allows inexact matches using magic characters

For example:

getHIDsbyDescription("Network*") -> Will return all the HIDs with description starting with Network getHIDsbyDescription(*Peter'sPortable*") -> Will return all the HIDs with description containing Peter'sPortable

Parameters:

*description* The description to match against

Returns:

A **java.lang.String[]** containing all the HIDs in the LinkSmart P2P **network** that match the description

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHostHIDsbyDescriptionAsString (String *description*) throws RemoteException

Operation to retrieve all HIDs in the LinkSmart P2P **network** (in String format) that match a description. The description allows inexact matches using magic characters

For example:

getHIDsbyDescription("Network*") -> Will return all the HIDs with description starting with Network getHIDsbyDescription(*Peter'sPortable*") -> Will return all the HIDs with description containing Peter'sPortable

Parameters:

*description* The description to match against

Returns:

A **String** containing all the HIDs in the LinkSmart P2P **network** that match the description (separated by commas)

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getDescriptionbyHID (String *hid*) throws RemoteException

Method to retrieve the description associated with a given hid.

Parameters:

*hid* The hid

Returns:

A **String** with the description associated with the requested hid

| | |
|---|---|
| | String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getIPbyHID (String *hid*) throws RemoteException |
| | Method to retrieve the ip associated with a given hid. |
| | Parameters: |
| | *hid* The hid |
| | Returns: |
| | A **String** with the ip associated with the requested hid |
| | String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getEndpointbyHID (String *hid*) throws RemoteException |
| | Method to retrieve the endpoint associated with a given hid. |
| | Parameters: |
| | *hid* The hid |
| | Returns: |
| | A **String** with the endpoint associated with the requested hid |
| | String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHostHIDEndpoint (String *hid*) throws RemoteException |
| | Get the Host HID endpoint |
| | Parameters: |
| | *hid* the HID |
| | Returns: |
| | the Host HID endpoint |
| | Deprecated: |
| | void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.removeHID (String *hid*) |
| | Operation to remove an HID from the Network Manager |
| | Parameters: |
| | *hid* the HID to remove Network |
| | void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.removeAllHID () |
| | Removes all HIDs Deprecated: |
| | String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getNMPosition () throws RemoteException |
| | Gets the NM position |
| | Returns: |
| | last know NM position |
| | Exceptions: |
| | *RemoteException* |
| | Deprecated: |
| | String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getNMPositionAuth (String *in0*) throws RemoteException |
| | Parameters: |
| | *in0* deprecated |
| | Exceptions: |
| | *RemoteException* |
| | Returns: |
| | the NM position auth |
| | Deprecated: |
| | String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getInformatio |

nAssociatedWithHID (String *senderHID*, String *receiverHID*)

This method exchanges certificates between two HIDs.

As a result of this method, two entries will be added to the CryptoManager's keystore:

1. the certificate of receiverHID 2. a symmetric key that can be used by the Inside LinkSmart module for subsequent communication. The certificate that has been stored in the CryptoManager can be referenced using the return value of this method.

Parameters:

| | |
|---|---|
| *senderHID* | Your own HID. |
| *receiverHID* | The target's HID. |

Returns:

a String, representing different Attributes that could be retrieved from the receiverHID's certificate.

String [] eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHIDByAttributes (String *requesterHID*, String *requesterAttributes*, String *query*, long *maxTime*, int *maxHIDs*)

This method searches the LinkSmart Network for the HIDs that contain attributes matching the query. The format of the queries is as follows:

(key1=cond1)&&(key2=cond2*)...

The developer can also use the magic character (*) for inexact queries. Due to this, the number of HIDs matching the query might be greater than one. Thus, the developer can specify the number of HIDs that will be returned and the maximum time to wait for resolving the query. For example, if maxTime = 60000 ms and maxHIDs = 4 the result of the query will be returned when the Network Manager gets 4 (or more) HIDs that match the query or when maxTime expires.

The allowed values for maxTime are 0 (search only locally) - 60000 (ms) and for maxHIDs 0 (best effort) - maxInt *

Parameters:

| | |
|---|---|
| *requesterHID* | Your own HID. |
| *requesterAttributes* | The target's HID. |
| *query* | Query for HID attributes: |

`(key1=cond1) && (key2=cond2*)...` * = non-exact match

| | |
|---|---|
| *maxTime* | Maximum time to wait for query resolution in ms |
| *maxHIDs* | Maximum number of HIDs to return |

Returns:

a `String`, with the resulting HID separated by blank spaces (0.0.0.1 0.0.0.2 ...)

String eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.getHIDByAttributesAsString (String *requesterHID*, String *requesterAttributes*, String *query*, long *maxTime*, int *maxHIDs*)

This method searches the LinkSmart Network for the HIDs that contain attributes matching the query. The format of the queries is as follows:

`(key1=cond1)&&(key2=cond2*)...`

The developer can also use the magic character (*) for inexact queries. Due to this, the number of HIDs matching the query might be greater than one. Thus, the developer can specify the number of HIDs that will be returned and the maximum time to wait for resolving the query. For example, if maxTime = 60000 ms and maxHIDs = 4 the result of the query will be returned when the Network Manager gets 4 (or more) HIDs that match the query or when maxTime expires.

The allowed values for maxTime are 0 (search only locally) - 60000 (ms) and for maxHIDs 0 (best effort) - maxInt *

Parameters:

| | |
|---|---|
| *requesterHID* | Your own HID. |
| *requesterAttributes* | The target's HID. |
| *query* | Query for HID attributes: |

`(key1=cond1) && (key2=cond2*)...` * = non-exact match

| | |
|---|---|
| *maxTime* | Maximum time to wait for query resolution in ms |

| | |
|---|---|
| | *maxHIDs* Maximum number of HIDs to return<br>Returns:<br>a `String`, with the resulting HID separated by blank spaces (0.0.0.1 0.0.0.2 ...)<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.setTrustThreshold (double *trustth*) `[protected]`<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.setTrustManager (String *url*) `[protected]`<br>void eu.linksmart.network.impl.NetworkManagerApplicationSoapBindingImpl.createSecureSession () `[private]` |

Table 10 Performance and Fault Management

| | | |
|---|---|---|
| Main functions: | • Classify traffic and track network traffic usage<br>• Track network bandwidth utilization<br>• Identify performance and security issues | |
| Processed requirements: | REACTION-25 | Fault tolerance to network malfunctioning. |
| | REACTION-60 | Restore from malfunctioning. |
| | REACTION-134 | Any interface between an end-user and the platform shall have a reasonable maximum response time in condition of public network optimally working. |
| | REACTION-136 | The platform shall cater for 20 simultaneous users in the field trials. In the end product this number is expected to grow to 100. |
| | REACTION-165 | Error Messages. |
| Components | Traffic Sniffer, Traffic Analyzer, Report Engine, Db Storage | |
| Dependencies | n/a | |
| Interface | n/a | |

Table 11 Network Monitoring

| | | |
|---|---|---|
| Main functions: | • The first functional area involves the initialization of the android devices in the REACTION network. This entails an automatic configuration process which is controlled by the REACTION backend (an Edge Monitoring Node, or EMN, to be precise) and during which the device obtains information that is necessary to start communicating with the REACTION backend, e.g. a unique identifier, a description of the information to be periodically transmitted, etc.<br>• The second area involves the action of broadcasting status updates by the android devices operated either at the patient's premises or the hospital wards.<br>• The last functional area is the periodic polling from the monitoring server in order to collect the status for the various devices in the network. Each function includes complementary actions, such as retrieving the status of each device from the underlying OS, storing the status in a database at the monitoring server, etc. | |
| Processed requirements: | REACTION-18 | Monitoring devices must be discoverable by existing network infrastructure. |
| | REACTION-358 | Network manager for hosting client. |

| | | |
|---|---|---|
| | REACTION-54 | Network & system monitoring. |
| | REACTION-89 | Network management subsets. |
| | REACTION-94 | Availability: Patient data and other resources must be available to ensure proper treatment. |
| | REACTION-127 | Home and mobile gateway. |
| | REACTION-168 | Remote Patient Monitoring. |
| Components | Android devices (Application Hosting Devices - AHDs), Edge Monitoring Nodes (EMNs), A Central Monitoring Node (CMN) | |
| Dependencies | n/a | |
| Interface | n/a | |

Table 12 Security Manager

| | | |
|---|---|---|
| Main functions: | • Authenticate users of components<br>• Decrypt incoming messages / encrypt outgoing messages, if necessary<br>• Control access to components | |
| | REACTION-45 | Protection against threats. |
| | REACTION-91 | Authenticity: Processors of information should be able to determine whether the data being processed is authentic. |
| | REACTION-92 | Integrity: Information, in particular health data, must be protected from any kind of unintended changes during transport. |
| | REACTION-100 | Access control: Access to sensitive information should only by given to authorised personnel. |
| | REACTION-109 | Scalability: the security must not materially impact the performance of the system. |
| | REACTION-115 | Transparency: Security configuration should be hidden from the user as far as possible. |
| | REACTION-116 | Availability of security mechanisms to manage sensitive data. |
| | REACTION-118 | Assurance: the architecture and its implementation must provide assurance that it delivers the security and compliance properties it promises. |
| | REACTION-145 | The user must consent to the collection of personal data whenever possible. > moved from Data Collection Manager! |
| | REACTION-325 | The possibility to manage user's account by username and password and secure log in and log out. |
| | REACTION-339 | Communication between the Reaction Device Hosting Server and the patient's/GP's web browser MUST be authentic (entity authentication), with integrity, and confidential. |
| | REACTION-385 | Digital identities for the Reaction platform MUST only be issued or revoked by trusted (third) parties, e.g., a certification authority (CA). |
| Components | Library (Jar), Framework Module, Web Service Proxy, TLS engine | |
| Dependencies | Developer of Component, Web Service Framework being used, Trust Model employed for component hosts, Web Server type | |

| | |
|---|---|
| Interface | n/a |

Table 13 Identity Manager

| | | |
|---|---|---|
| Main functions: | • Create new user.<br>• Update user.<br>• Delete user.<br>• Assign role.<br>• Add password.<br>• Change password. | |
| Processed requirements: | REACTION-35 | Usage Data (Information about elder and juvenile usage of the platform and resources shall be available). |
| | REACTION-44 | Protection against unintended user actions. |
| | REACTION-90 | Identifiability: Recipients and senders of information must be identifiable, though not necessarily personally identifiable. |
| | REACTION-93 | Confidentiality: Sensitive information must not be readable by unauthorised persons. |
| | REACTION-99 | Authorisation: Stakeholders must be authorised before they are allowed to perform relevant actions. |
| | REACTION-175 | Automated identification of users (caregivers) working with REACTION front-end in the hospital. |
| | REACTION-197 | Care spaces in the outpatient environment. |
| | REACTION-246 | Multi-user availability and display of the fever chart. |
| | REACTION-326 | The registration of the enrolled patient on to the system shall be accured manually by the Care giver at the Primary Care. |
| | REACTION-336 | Patient enrolment (or recruitment). |
| | REACTION-341 | Roles MUST be defined for stakeholders of the Reaction platform, e.g., doctor, nurse, patient, informal carer, administrative personnel etc. |
| | REACTION-343 | Every person represented in the Reaction platform MUST be assigned to one or more roles. |
| | REACTION-376 | Integrity check for the stored data. |
| | REACTION-415 | Each person MAY only perform actions permitted by her role. |
| | REACTION-437 | Each role MUST be assigned to a set of permissible actions. |
| Components | n/a | |
| Dependencies | Security Manager, HID Manager Component | |
| Interface | n/a | |

Table 14 IoTDevice

| | |
|---|---|
| Main functions: | • Maps requests to device services<br>• Generates responses<br>• Advertising IoT device descriptions |

| | | |
|---|---|---|
| | • Advertises device services | |
| Processed requirements: | REACTION-6 | Any REACTION device should have an associated semantic model (description). |
| | REACTION-125 | Portable device should collect also additional environmental measurements. |
| | REACTION-334 | Devices should be able to operate anywhere in the home. |
| | REACTION-401 | Device specialization. |
| Components | Library (Jar), Framework Module, Web Service Proxy, TLS engine | |
| Dependencies | IoT Device Manager | |
| Interface | System.String GetIoTID ()<br>Returns the IoTID for the device.<br>**Returns:**<br>The HID of the device<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>System.String GetStatus ()<br>Returns the status for the device.<br>**Returns:**<br>The status of the device<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>System.String GetProperty (System.String *Property*)<br>Returns a property of the device. A developer can choose any properties he like to use and set.<br>Parameters:<br>*property* A valid property name (valid XML element name)<br>**Returns:**<br>The value of the property<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>System.Boolean GetHasError ()<br>Tells if the device has an error.<br>**Returns:**<br>True if error, false otherwise<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>System.String GetErrorMessage ()<br>Returns an errormessage for the device.<br>**Returns:**<br>Latest errormessage<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>System.String GetPhysicalDiscoveryInfo ()<br>Returns the physical discovery information that is associated with the device.<br>**Returns:**<br>An XML string<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>System.String GetIoTDeviceXML ()<br>Returns the device model XML for the **IoT** Device.<br>**Returns:**<br>the device model XML in SCPD format<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>System.String GetDACEndpoint ()<br>Returns the endpoint of the DAC that has discovered the device. | |

| | |
|---|---|
| | **Returns:**<br>The endpoint of the current DAC<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>void SetIoTID (System.String *IoTID*)<br>Sets the IoTID for the device.<br><u>Parameters</u>:<br>*IoTID* The valid **IoT** ID<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>void SetStatus (System.String *Status*)<br>Sets the status for the device.<br><u>Parameters</u>:<br>*Status* A status value choosen by the developer<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>void SetProperty (System.String *Property*, System.String *Value*<br>Sets a property of the device. A developer can choose any properties he like to use and set.<br><u>Parameters</u>:<br>*property* A valid property name (valid XML element name)<br>*value* The value of the property<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>void StartDevice ()<br>Starts the device.<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** .<br>void StopDevice ()<br>Stops the device, which will cause it to be removed from its current DAC. The device can later be re-started.<br>Implements **IoTWCFServiceLibrary::IIoTDeviceWSService** . |

Table 15 Medical Device

| | | |
|---|---|---|
| Main functions: | • Receive measurement from physical device<br>• Send measurement to server side or other modules<br>• Receive events from physical device<br>• Propagate events to Event Manager<br>• Get observation as plain ORU or XML ORU | |
| Processed requirements: | REACTION-3 | Support for IEEE medical device standards. |
| | REACTION-124 | Portable device should collect all the relevant vital signs measured on the patient. |
| | REACTION-180 | Measurement of glucose should be specific and the glucose sensor should be able to monitor glucose in complex media. |
| | REACTION-207 | ePatch communication. |
| Components | Library (Jar), Framework Module, Web Service Proxy, TLS engine | |
| Dependencies | IoT Device Manager, Orchestration Manager. | |
| Interface | ***System.String GetLatestObservationTime()***<br>Returns a timestamp of the latest stored observation<br><br>***System.String GetListOfObservations(System.Int32 numberOfObservations)***<br>Returns a collection of observation. The quantity of observations it set by the paramete<br>numberOfObservation. | |

| | |
|---|---|
| | ***System.String GetMsh()***<br>Returns the MSH segment set for the latest HL7 ORUR01 message.<br><br>***System.String GetObservation()***<br>Returns the observation made in HL7 ORUR01 format.<br><br>***System.String GetObservationXml()***<br>Returns the observation made in XML format.<br><br>***System.String GetPatientID()***<br>Returns the PID segment set for the latest HL7 ORUR01 message.<br><br>***void SetMsh(System.String msh)***<br>Manually set the MSH segment with the parameter msh. XML format support only.<br><br>***void SetPatientID(System.String patientID)***<br>Manually set the PID segment with parameter patientID. XML format support only.<br>***void SetEquipmentID(System.String equipmentID)***<br>Set the equipment ID given from the device. Should be presented in EUI-64 format.<br><br>***void SetPatientIdentifier(System.String idNumber, System.String checkDigit, System.String checkDigitScheme, System.String assignAuthority, System.String identifierTypeCode)***<br>Populate the patient identifier field in the PID segment.<br><br>***void SetPatientName(System.String familyName, System.String givenName, System.String nameTypeCode)***<br>Populate the patient name field in the PID segment<br><br>***void SetReceivingApplication(System.String namespaceID, System.String universalID, System.String universalIDType)***<br>Populate the receiving application field in MSH segment<br><br>***void SetReceivingFacility(System.String namespaceID, System.String universalID, System.String universalIDType)***<br>Populate the receiving facility field in MSH segment<br><br>***void SetSendingApplication(System.String namespaceID, System.String universalID, System.String universalIDType)***<br>Populate the sending application field in MSH segment<br><br>***void SetSendingFacility(System.String namespaceID, System.String universalID, System.String universalIDType)***<br>Populate the sending facility field in MSH segment |

Table 16 REACTION Test Suite

| | | |
|---|---|---|
| Main functions: | • Create Medical Devices<br>• Set a new measurement<br>• Invoke Medical Device to send a HL7 ORUR01 message<br>• Test the Medical Device software functionality<br>• Test server side components that handle | |
| Processed requirements: | n/a | n/a |
| Components | Microsoft Silverlight | |
| Dependencies | Device Connetivity Kit, MeasurementWS | |

| | |
|---|---|
| Interface | n/a |

Table 17 Data Fusion Engine

| | | |
|---|---|---|
| Main functions: | • Set data fusion scheme<br>• Discovery and connect to sub devices<br>• Collect data<br>• Listen to device events<br>• Aggregate and correlate events and observations based on data fusion scheme. | |
| Processed requirements: | REACTION-342 | Low-level data fusion. |
| | REACTION-365 | Data should be stored in proper way in order to be easily transmitted over mobile networks in case that broadband network is not available. |
| | REACTION-454 | Content formatter. |
| Components | n/a | |
| Dependencies | IoT Device, Medical Device, Event Manager. | |
| Interface | `void AddFusedDevice(System.String devicename, System.String devicexpath);`<br><br>`System.String GetDataFusionScheme();`<br><br>`System.String GetFusedValue(string name);`<br><br>`System.Int32 GetNumberOfFusedDevices();`<br><br>`void RemoveFusedDevice(System.String devicename);`<br><br>`void SetDataFusionScheme(System.String datafusionscheme);` | |

Table 18 Long-term Risk Manager

| | | |
|---|---|---|
| Main functions: | • Accept a **risk-assessment request** as an input: the request should contain the patient profile and the diabetes complication(s) of interest<br>• Process the patient data and compute the risk profile, even when some measurements of the clinical parameters are not provided in the patient profile (**missing information**)<br>• Return the risk-profile as a function of the personalized risk and probability for developing the complication **over time** | |
| Processed requirements: | REACTION-81 | Long-term risk calculation and patient-oriented presentation. |
| Components | Service Core | |
| Dependencies | n/a | |
| Interface | n/a | |

Table 19 Short-term Risk Manager

| | |
|---|---|
| Main functions: | • identify blood glucose trends and patterns |

| | | |
|---|---|---|
| | • short-term risk predictions concerning the complications | |
| Processed requirements: | REACTION-83 | Interface to clinical data from "near" real-time observations for decision support. |
| Components | Data analysis, Short-term risk models, | |
| Dependencies | n/a | |
| Interface | n/a | |

Table 20 Risk Classification Manager

| | | |
|---|---|---|
| Main functions: | • Assist in defining risk class: low/medium/elevated/high | |
| Processed requirements: | REACTION-86 | Estimate short- and mid-term risk and identify successful therapy schemes for patient groups. |
| | REACTION-153 | Symptoms of diabetes or hyperglycaemia. |
| | REACTION-184 | Risk values for HbA1c. |
| | REACTION-241 | Management of hypoglycaemic episodes in Inpatient environment. |
| | REACTION-409 | Risk assessment models and rules. |
| | REACTION-465 | Clinical evaluation report. |
| Components | Risk classification engine, Classification model | |
| Dependencies | This component uses the outputs of the Data Integration Module and the Care Plan. | |
| Interface | n/a | |

Table 21 Generic Decision Support Development Component

| | | |
|---|---|---|
| Main functions: | • An interface which supports the specification of the DS module to be developed, including:<br>o the selection of the knowledge representation type to be used (e.g. rules, decision tree, guideline) and the definition of the knowledge necessary for the decision making<br>o the definition of the necessary input parameters (data items), e.g. it can use the outputs of the Data Integration Module<br>o the definition of the expected outputs – it can be e.g. advice, diagnosis, prediction<br>o the selection of the user interface to be used by the DS module<br>• Data structures for the above elements<br>• A generator for the construction of the DS module from the above inputs | |
| Processed requirements: | REACTION-226 | Electronic fever/sugar chart should be modelled in the data management system. |
| | REACTION-227 | Initialization of the fever/sugar chart. |
| | REACTION-251 | Creation of electronic decision support rules shall be supported. |
| | REACTION-391 | Data fields for the inpatient glucose control prototype (eDSS). |
| | REACTION-466 | (Web) Service to present decision support for glucose |

| | | |
|---|---|---|
| | | control to clinicians. |
| Components | Knowledge base, User interface, DS module specification, module generator, decision support module. | |
| Dependencies | n/a | |
| Interface | n/a | |

Table 22 Generic Questionnaire Development component

| | | |
|---|---|---|
| Main functions: | • Interface for the definition of the type of the questionnaire to be developed. It allows the definition of questions and their relationships, potential answers and their type.<br>• Interface for the definition of the text generation. It allows the definition of the text generation rules and the structure of the report. (E.g. it permits to define the classification of the questions by their importance).<br>• Questionnaire database for the storage of the above defined questionnaire elements.<br>• Text generation database for the storage of the above defined components<br>• Generic user interface to support the questionnaire development process (collection of the necessary data, generation of the structure of the questionnaire, activation of the questionnaire interface and its service programs, actualisation/fulfilment of the questionnaire, testing).<br>• Generic user interface to support the report generator development process (collection of the necessary data, development of the report generator structure, activation of the report generator interface and its service programs, actualisation/fulfilment of the report generator, testing). | |
| Processed requirements: | REACTION-126 | Portable device should allow patients to complete the acquired data set with questionnaire or additional information (status, activity, food intake). |
| | REACTION-330 | Ability for the patient to have an access to a library of diseases with a list of questionnaires which help the patient to manage his/her lifestyle and disease in a better way. |
| | REACTION-349 | Patient questionnaires (lifestyle, physio-psychological conditions, checking medication compliance, adherence to clinical pathways, education, self-management). |
| Components | Questionnaire definition interface, Report definition interface, Questionnaire database, Text generation database, Questionnaire development interface, Report development interface. | |
| Dependencies | The data collected by the questionnaires will be managed by a special Qualitative Data Management Module (QDMM), which collects and integrates the reports of the selected questionnaires. | |
| Interface | n/a | |

Table 23 Interface adapters

| | | |
|---|---|---|
| Main functions: | • Providing interfaces to access internal existing information systems<br>• Providing interfaces to access and communicate with external systems and services, such as cloud-based services. | |
| Processed requirements: | REACTION-32 | The architecture should support the Continua WAN interface (WAN-IF). |
| | REACTION-84 | Interface to patients' health history information from EPR/HIS. |

| | | |
|---|---|---|
| | REACTION-362 | Interface to patient demographic register. |
| | REACTION-363 | Interface to Hospital Information System for clinical data import/export. |
| | REACTION-400 | Data/messages exchanged between the Reaction Device Hosting Server and the EPR/EHR System SHOULD be authentic (message authentication), with integrity, and confidential. |
| | REACTION-413 | Connection with external services like MS HealthVault and RunKeeper |
| | REACTION-431 | Data/messages exchanged between the Reaction Device Hosting Server and the GP EPR SHOULD be authentic (message authentication), with integrity, and confidential. |
| | REACTION-434 | Interface to Lab Information System (LIS) for glucose data import. |
| Components | n/a | |
| Dependencies | n/a | |
| Interface | n/a | |