MEDOLUTION

D2.3 Medolution Best Practice Architecture
Work package 2: architecture and system integration

Project number:	**14003 Medolution**
Document version no. WP2 – version 1.0	
Edited by:	Jeroen Sonnemans, Philips
Status:	Final

ITEA Roadmap domains:
Major: Content & Knowledge

ITEA Roadmap categories:
Major: Interaction
Minor: Network & computing

HISTORY

Document version #	Date-	Remarks
V1.0	2018-09-21	Final deliverable

TABLE OF CONTENTS

1 Introduction

1.1 Purpose of this document

The purpose of this document is to provide a "Best Practice" architectural view of the system concepts and demonstrators developed during the Medolution project. The resulting architecture can be used to provide a common view of Medolution as a "System" to which the individual use cases can be mapped. This architecture can also be used as recommendation for future projects that aim at further developing Medolution-like systems.

This document is structured according to The Open Group Architecture Framework (TOGAF). TOGAF provides an iterative Architectural Development Method (ADM) that is shown in Figure 1.

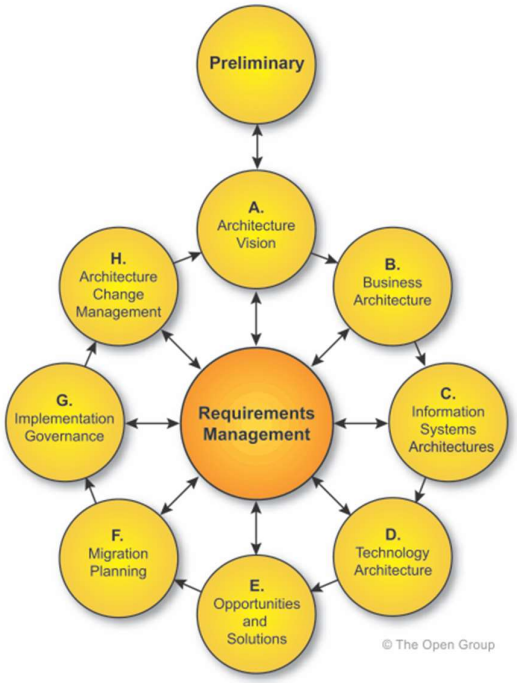


Figure 1: The Architecture Development Methodology' of TOGAF

As such, this document can be seen as the converged outcome of several iterations of ADM containing the results of

- Phase A – Architecture Vision
- Phase B – Business Architecture
- Phase C – Information Systems Architectures
- Phase D – Technology Architecture

The scope of the business architecture is to provide an overview of business models and involved stakeholders that are applicable when commercializing a Medolution-like system.

The scope of the information systems architecture is to provide an overview of security and privacy standards applicable to Medolution-like systems once they would be commercialized in relation to the types of data stored, shared and transferred. These data types consist of non-personal, personal, and medical data. Achieving compliance to these standards is the main architectural driver behind information systems architecture in Medolution-like systems.

The scope of the technology architecture is to provide an abstract functional architectural view that is based on the detailed technology architectures of individual Medolution demonstrators. This provides a generic best-practice decomposition of a Medolution system into functional components. Individual demonstrators use (a sub-set of) concrete implementations of these functional components. From this decomposition key architectural best practices are derived. Besides this decomposition, the technology architecture provides a blueprint on how to incorporate dependability analysis and related design in critical parts of the Medolution environment. This ranges from a local device/sensor level to cloud level.

Phases E to H are out of scope of this document.

1.2 Related Deliverables

The following deliverables are related to or contain already material and relevant input with regard to the Medolution architecture:

Deliverables
D1.1 – State of the Art Analysis
D3.2-3.4-3.5 Guideline for development of Big Dependable System components to enable short time to market interoperability between medical devices and BDSs + Software release
D4.4 – Medolution Platform APIs and Specification V2

2 Architecture Vision

The landscape of healthcare is changing rapidly: worldwide, the population as a whole is ageing and at the same time people (patients) live longer, often with one or more chronical diseases, causing structural increased cost of healthcare. In combination with these trends, the digitalization of healthcare develops at high pace and intensity, resulting in large amounts of heterogeneous, clinically relevant information, in great technical variety, from many sources, for health professionals and patients, however scattered all over the place. All this information needs in fact to be holistically handled,synchronically managed and accessiblein real time. This is a fundamental challenge not only for care providers and public authorities but also for patients. This is the challenge the Medolution architecture addresses.
Medolution is in fact an ”information system blueprint” that can be economically and technically instantiated in many ways. There is not only “one road to Rome”, however, any system addressing the above-mentioned challenges is expected to adress these challenges using the concepts presented in this architecture.

A Medolution system description is not restricted to technical solution descriptions but includes a definition of the business aspects as well as its functional characteristics.
The vision is that this Medolution concept will deliver the “information system concept” that brings the relevant medical information to health professionals and patients at the right time at the right place, in the most effective, and intelligent way, constructed from a variety of sources of different modality, within the economical context and reality of the business of health care institutions (hospitals and doctors) and patients (insurance companies, government). Unlike limitations of the existing point-to-point capabilities and applications, Medolution should allow for scaling to millions of patients in parallel, supporting information flows from a multitude of sensor devices and health applications towards many specialized clinical applications. **Medolution should deliver the methods to seamlessly connect these applications for the end-user; it should not be restricted to one disease, and should address disease as a generic malfunction, thus enabling the serving of any number of patients and clinicians.**
Systems based on the Medolution architecture are meant to be trustworthy sociatal infrastructure services, dedicated to healthcare professionals. To realize this, Medolution will use the concept of so called “Smart Patient Environments”. These are in the heart of the vision.
The scope of Medolution patient and hospital “data” streams should span the entire “continuum of care’’ as illustrated in Figure 2.

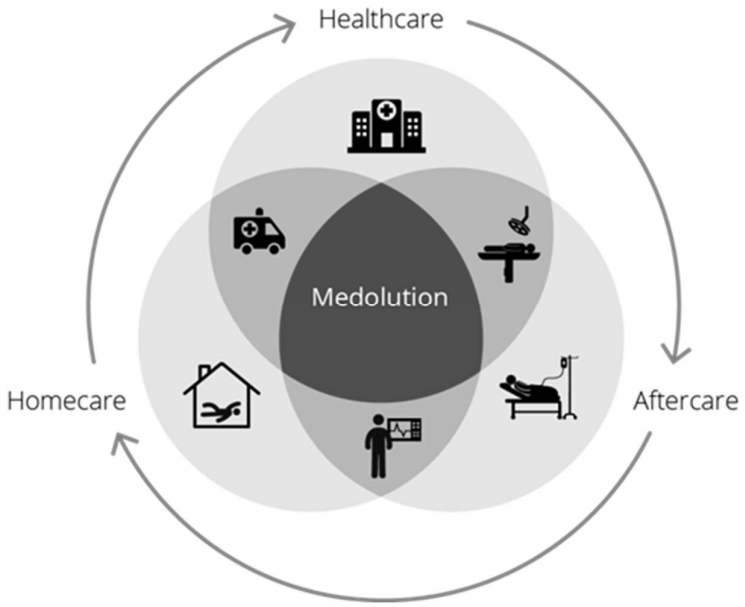


Figure 2: Continuum of care

A "Smart Patient Environment" consists of multiple automated devices (such as vital sign sensors or an artificial heart pump, for example) and a variety of systems (such as protocol management tools, EHR systems, diagnostic tools and data analytics applications), locally and remotely connected over networks, that produce and deliver data, continuously monitored and automatically interpreted on the basis of personalized medical protocols for medical "alerting" and clinical decision support. This way, Medolution will enable continuous and intermittent monitoring of patients in emergency situations, as well as patients with chronic diseases who are equipped with data gathering devices (body sensors, medical devices, environment sensors, video, sound devices, some of which may have rule based automated or manual remote control). These devices (be them professional or personal) continuously transmit their data over a secure and reliable network for *professional medical* analysis and decision support. These data will instantly and constantly be analyzed against predictive models, including their cumulative effects, taking the reliability of the data source and traffic into account. Decision support includes the application of protocol- and evidence based algorithms as configured per single patient by the medical professional. Personalized medical protocols will take individual conditions into account.

Not only chronical disease patients should be included in a Medolution system, but also acute and sub-acute situations, both inside and outside hospitals. Acute situations are time sensitive: every minute earlier treatment in an acute situation – for instance a stroke or a heart attack – has a direct impact on the downstream effects: not only quality of life improves, but also more efficient use of hospital resources and a lower volume of intramural care can be realized.

Analysis and decision support is configurable and scalable to handle millions of patients in parallel. The Medolution concept will thus go far beyond the hundreds presently existing point-to-point solutions.

3 Business Architecture

The Business Layer of the architecture focuses on the business and organizational aspects that a Medolution system "offers". It describes the business concept – how to make it economically viable -, the stakeholders, the roles and players, the buyers, the users, the offering parties. The Business Layer Architecture describes the business options that enable the execution of the Medolution concept to function in the market. The Medolution functional and technical requirements and specifications should be impacted by the Business Layer of the Architecture.

3.1 Medolution as a Service

Most probably the Medolution concept is materialized as a "commercial" service or "ecosystem of services", that can only be used by end users against a kind of payment. The payment to subscribe to the service can have different formats – pay per usage, pay per time period, pay per volume of data, pay for speed, pay for specific security services, pay for a certain algorithm, etc.

3.2 The Medolution Offering Side

The question is of course "who" would make the service available to the medical healthcare market as the Medolution service provider. We believe that we will see many modalities, and many players, whereby also hospitals will perhaps try to play that role. However, we believe that "connection of devices, systems and applications" typically is an IT business, not a medical business, and because of its intra- and extramural character cannot be delivered by hospital IT departments. It can neither be a national business, because patients travel. We assume that Medolution is a global operating service, using the Internet and Cloud facilities that can deliver the scale of operation Medolution requires. We will see competitive suppliers of the Medolution service, probably focusing on selling the service into their regions.
In this chapter, we imagine "a" company that could be such a "regional" Medolution service provider in and for Europe. We call it "MedolutionCo". It could be a real company, a virtual company, or an ecosystem of companies, each of which delivers "a service or a component thereof" against "a" payment; in that sense it could be like a market, in which a customer could shop the required services to construct his overall service.
There is a need to manage the Medolution service from a business perspective, from medical perspective and from a technical perspective as well:

- Business perspective: marketing and sales, customer contracting, transaction management, usage control and monitoring, liability management, applying government regulations, organizing data access permissions, agreements with insurance companies

- Medical perspective: MedolutionCo needs to run a protocol management system – basis for regulated (=permitted) diagnosis, treatment advice and decision support
- Technical perspective: a MedolutionCo system has to be dependable, meaning to deliver a 24/7, high performance infrastructure and the application services that run on it: front office apps as well as back office apps.

We assume that MedolutionCo has a European IT development center and a multilingual support center/helpdesk. The service delivers a "medical virtual collaboration infrastructure for acute and chronical medical events" – "whereby the life of a patient is in the center" – It includes capabilities like:

- Licensees - Participants can talk to each other while sharing live streamed images, live video streams , data from patient sensors, patient history, and local observations (examinations) of the patient
- Participants can "annotate" the "event" with discussions, ideas, final diagnostics, treatment proposals, that are stored for re-use
- Participants are supported by the relevant (international) protocols and protocol based decision support. Protocols are selected after first global diagnosis.
- High Performance: sound, images and any other type of information should be delivered within the limits of accepted latency and the prescribed quality level
- We can imagine less and more luxury services – performance levels (bandwidth), with or without annotation, with or without decision support.
- The service can be stopped and then restarted without loss of information. MedolutionCo should be able to guarantee this or have a warranty in place.

3.3 Guaranteed Confidentiality of Doctor and Patient Data

At the offering side the guarantee of confidentiality is a critical factor. MedolutionCo. will, in many cases, use 3rd party services to support its offer, so it needs to acquire the rights and licenses to use these services. At the same time another important point of attention is of course that MedolutionCo as a legal entity has to guarantee that the data of its customers are treated with full confidentiality and as secure as possible. This should be managed carefully. Chapter 4 will provide an overview of applicable standards that currently apply to information systems handling both personal and medical data. To handle these factors, the MedolutionCo ecosystem needs to contain multiple administrative registers in its back-office, mapping identity of users among and between different, distributed application domains, where these domains can have their own proper external authorization. For instance, an ECG service provider needs to ensure that data of specific patients can only be accessed by specific physicians.
In the medical domain it is critical that only approved individuals can see the data of a patient. Additionally, it is critical for any recorded data from any data source in the MedolutionCo ecosystem to be mapped to the correct individual (patient). This in itself

is not a new challenge and it is well described in for example the IHE IT Infrastructure Profiles. However in a distributed, international setting not easy to realize.
A deployment of a solution using the MedolutionCo ecosystem needs to comply with regulations and guidelines which are specific to each country. For example, how medical professionals identify themselves in Europe is defined by guidelines from the European commission, subsequently translated into laws by the European countries. In Europe, all medical professionals have a smart card which can be used in combination with other security measures to ensure a secure and trusted authentication.

As the MedolutionCo ecosystem is not limited to a single country, its service need to be able to bridge the country regulated authentication mechanisms to the Medolution-specific identification and authorization tools. Authentication of individuals can be done by parties outside Medolution. However, according to the European legal situation, authorization should be based on identification and needs to be part of any service exposing medical data. Mapping of identities between various systems and authentication providers has to be part of the MedolutionCo service.

An important aspect of Confidentiality is the "security" component. MedolutionCo will have to invest resources to meet requirements like the following:

- Data in transfer has to be always encrypted
- Certificate based security between devices, gateways, and the cloud has to be provided.
- The legal constraints may require that data is anonymized
- The integration of hospital information system data has to be possible but security and privacy constraints need to be established.
- Anonymization has to be made available on request.
- Data must always contain timestamps for ordering
- Data semantics must be unambiguous – which is not easy in an international distributed system.

Consequently MedolutionCo will be a contract-intensive company and will invest heavily to protect its liabilities. Therefore we expect that a consortium of companies and public-private organizations will cooperate to bring MedolutionCo to live. The technical feasibility is there, the business feasibility is however not easy, but a must go route for governments. Governments are under political pressure to maintain a certain health care level for their populations. So, they will probably create the legal conditions for the business to start the MedolutionCo service in a more or less protected manner.

3.4 The Medolution Buyer Side

The medical "market" can subscribe to the MedolutionCo offered service. For example:

- a network of care organizations that regularly have to collaborate around patients
- a large hospital that subscribes and is allowed to give access to the service to 3rd parties under given conditions
- a department of a hospital can subscribe
- a group of GPs
- an insurance company that subscribes to the service and allows its customers to use it.
- a government that subscribes and makes it available to the medical networks in its city, region or country
- a neutral foundation that combines government, insurance and the medical sector in a region

Once a "customer" has signed the MedolutionCo contract, the MedolutionCo team needs to setup the infrastructure for this customer:

- The MedolutionCo service needs access to data sources from hospitals, the GPs, the patient
- The service needs to have working APIs to extract and see data from these sources; sources include: imaging systems (PACS), EHR systems, medication Systems, sensors put on the patient's body
- The MedolutionCo service back office holds the functional and technical documentation – transparency of how it works, data security, etc. that needs to be transparent to the customers
- MedolutionCo will train users how to use the service
- The service has a functional and technical support helpdesk, 24/7

3.5 Conclusion of the chapter

MedolutionCo needs legal conditions from governments that allow it to work in a protected manner. Without these conditions fulfilled, the commercial market will hesitate to take responsibility. We expect therefore that public-private institutions and foundations need to be created to take advantage of the new technical capabilities, for the health of their population and the global competitiveness of its medical sector.
This business layer of the architecture will find further translation in the following chapters, both from an information management and technical perspective. It will be illustrated how different use cases materialize the Medolution information concept at a small exemplary scale. It will make clear that MedolutionCo still has a route to go.

4 Information Systems Architecture

4.1 Introduction

The main architectural driver for Information System Architecture in a Medolution-like system that stores personal and medical data of patients is to achieve compliance with applicable data privacy regulations and security standards. First a short introduction is given into two important data security regulations applicable to Medolution. Next we define the data categories as present within Medolution and explore these categories against relevant quality aspects. Technological means of complying to these standards are provided in ***D1.1*** – *State of the Art Analysis.*

4.2 Regulatory standards

4.2.1 General Data Protection Regulation

Data used within healthcare applications and solutions must be demonstrable compliant with applicable regulations. One example of such a regulation is the General Data Protection Regulation (GDPR), more precisely: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC.

The scope of health data in these regulations has to be interpreted broadly. Any data about the physical or mental health status or a data subject generated in a professional medical context must be considered health data.

This scope is further widened as it will include any information about 'disease risk'. So this refers to any data concerning the potential future health status. Whether the actual used device or software is a medical device in the context of the Medical Device Directive (MDD) is not relevant on deciding whether data is medical data in the sense of the GDPR and other regulations.

4.2.2 Data Protection Impact Assessment

Within the EU there are guidelines related to the GDPR to help assess data and impact of exposing data, these are part of article 29 of the GDPR. These guidelines help to execute a Data Protection Impact Assessment (DPIA) and determining whether processing is "likely to result in a high risk" for the purposes of Regulation 2016/679

The DPIA lists following nine criteria which would put the related processing in the high risk category: 1) Evaluation or scoring, 2) Automated-decision making with legal or similar significant effect, 3) Systematic monitoring, 4) Sensitive data or data of a highly personal nature, 5) Data processed on a large scale, 6) Matching or combining datasets, 7) Data concerning vulnerable data subjects, 8) Innovative use or applying new technological or organizational solutions and as last: 9) when the processing in itself "prevents data subjects from exercising a right or using a service or a contract"

It is quite evident that significant parts of Medolution data processing fit with multiple/most of these categories and hence all electronic personal health information (ePHI) within the Medolution systems falls into the high risk category.

So any organization that processes ePHI in the context of Medolution has the legal obligation to execute a DPIA and have processes and organization arranged to be able to deal with the related requirements from the GDPR. Figure 3 shows a schematic overview of a DPIA.

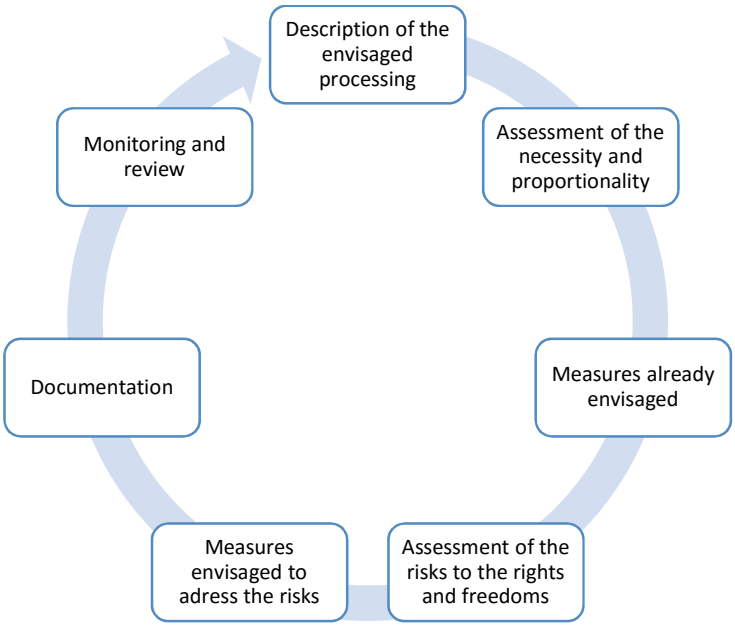


Figure 3: DPIA Process flow schematic

4.2.3 Health Insurance Portability and Accountability Act

The Health Insurance Portability and Accountability Act (HIPAA) is a Unites States legislation that provides data privacy and security provisions for safeguarding medical information. With HIPAA compliance (security aspects) it is mostly intended that the product is compliant with HIPAA Title II: HIPAA Administrative Simplification.

Title II directs the U.S. Department of Health and Human Services (HHS) to establish national standards for processing electronic healthcare transactions. It also requires healthcare organizations to implement secure electronic access to health data and to remain in compliance with privacy regulations set by HHS.

Main aspects covered in Title 2 are
National Provider Identifier Standard, Transactions and Code Sets Standard, HIPAA Privacy Rule(Officially known as the Standards for Privacy of Individually Identifiable Health Information), HIPAA Security Rule and the HIPAA Enforcement Rule.

4.2.4 NIST HIPAA Security Rule Toolkit

The NIST HIPAA Security Toolkit Application is a self-assessment survey intended to help organizations better understand the requirements of the HIPAA Security Rule (HSR), implement those requirements, and assess those implementations in their operational environment. A comprehensive user guide and instructions for using the application are available along with the HSR application.

4.2.5 Compliance

In order to be compliant in most jurisdictions for solutions in the Medolution context it is required to execute and act on the findings of both the DPIA and the NIST HIPAA Security Rule Toolkit. This compliance is based on current state of art and is NOT further detailed within the Medolution project.

4.3 Security standards

A significant number of security standards can be applied to systems as Medolution. We list a small set with relevant impact information storage and processing designs.

4.3.1 ISO/IEC 27001 and 27002

ISO/IEC 27001 formally specifies a management system that is intended to bring information security under explicit management control.

ISO/IEC 27002 defines good security management practice standard.

So ISO/IEC 27001 is relevant for structuring an organization to be able to handle data in a secure way and ISO/IEC 27002 provides concrete inputs into defining proper designs and architecture to ensure security.

4.3.2 NIST

The NIST Cybersecurity Framework (NIST CSF) "provides a high level taxonomy of cybersecurity outcomes and a methodology to assess and manage those outcomes." It is intended to help private sector organizations that provide critical infrastructure with guidance on how to protect it, along with relevant protections for privacy and civil liberties.

4.3.3 ISO 15408

This standard develops what is called the “Common Criteria”. It allows many different software and hardware products to be integrated and tested in a secure way.

4.4 Data Categories

At a high level we recognize devices that generate medical data, processes that analyze and transform data and potentially generate new data and applications that show data to a user of the system. These devices and processes additionally need to be controlled and monitored. This gives following classes of data:

First we identify data related to individuals within the Medolution context:

- Personal Medical Data, medical data explicitly related to a single individual.
- Personal NON Medical Data, NON Medical data explicitly related to a single individual.

This personal data can be anonymized creating

- Anonymized Medical Data, medical data explicitly NOT relatable to a single individual.
- Anonymized NON Medical Data, NON Medical data explicitly NOT relatable to a single individual.

Next we look at data related to the devices and systems within the Medolution context

- Service diagnostic data
- Service Management data
- Device management data
- Device diagnostic data

Data from specific devices can in theory be correlated to an individual using that device and could expose medical personal data. So for device diagnostic data and its processing an assessment is required to assess the risk of inadvertently expose personal medical data.

4.5 Mapping of data Categories to relevant Legislation and Standards

From previous Sections we have seen we have legislation related to Medical Data and Personal data, we have standards related to security of any data and more strict standards applicable to personal data and personal medical data.

Next table provides an overview of various legislation and standards and how/whether these apply to the defined data categories. If further specific assessment is required this is indicated as well.

Data category	Medical Data Legislation and standards	Personal Data Legislation	Medical Security standards	Security standards
Personal Medical Data	Yes	Yes	Yes	Yes
Personal NON Medical Data	No	Yes	No	Yes
Anonymized Medical Data	Assess	Assess	Assess	Yes
Anonymized NON Medical Data	No	Assess	No	Yes
Service diagnostic data	No		No	Yes
Service Management data	No		No	Yes
Device management data	Assess		Assess	Yes
Device diagnostic data	Assess		Assess	Yes

4.6 Data Security Measures

For all data it is obvious that it should only be accessible by authorized persons and services, so basic identity and access management (IAM) must be in place. Next all communications should be appropriately protected, e.g. by using secure communication (https) or virtual private networks. Stored data requires similar attention, depending on the nature of the data protected and/or encrypted storage is required.

These additionally relate to various dependability attributes:
Availability - readiness for correct service
Reliability - continuity of correct service
Safety - absence of catastrophic consequences on the user(s) and the environment
Integrity - absence of improper system alteration
Maintainability - ability for a process to undergo modifications and repairs, without loss of data
Confidentiality- the absence of unauthorized disclosure of information is also used when addressing security.
Veracity – integrity of data should be guaranteed and be verifiable.

Security is a composite of Confidentiality, Integrity, and Availability.

Practically, applying security measures to the components of a system improves the dependability by limiting the number of externally originated errors.

4.7 Best practice Information System principles

Within Medolution achieving compliance to privacy and security related legislations and standards is considered to be possible with state-of-the-art technology of which an

overview can be found in **D1.1** – *State of the Art Analysis*. Besides this the supercloud project can provide more detailed insights. See https://supercloud-project.eu/.

4.8 Conclusions of the chapter

Adherence to relevant security legislation requires data ownership and processesing by a legal entity, a corporation, or in some cases even the patient himself. In cases where data is processed by multiple partners it must be very clear and regulated through contracts who provides the service to the general public, persons, patients and has therefore to meet the related requirements from for example the GDPR.

Giving guidelines on handling data within the context of the Medolution project is depends on the type of data used in individual implementations of Medolution-like systems. Medolution use cases are examples of such individual system. Given the complexity of involved regulations and systems we can state a general way of working however it requires a competent organization to fulfill all the needs imposed by the applicable regulations. Companies involved in commercializing Medolution-like systems must have their own auditable quality systems which comply with these regulations and standards. In Cloud based systems a large portion of this compliance must be delegated to the cloud provider which a company selects to deploy its assets.

5 Technology Architecture

This chapter provides an overview of the general technology architecture that is applicable to all individual Medolution demonstrators. As indicated in the previous chapters, these demonstrators show the Medolution concepts in small individual examples. This provides a common technology view that covers all use cases. It can also be used as input for future projects that aim at building Medolution-like systems.

5.1 Functional view

Figure 4 shows a functional decomposition of Medolution system components and their dependencies. Within this decomposition 3 layers are distinguished:

- The (Medical) Device layer
- The Edge device layer
- The Cloud layer

The (Medical) device layer covers all measurement or treatment devices. Examples of such a device is the LVAD device or a blood pressure device. This includes hardware and embedded software running on the device. The local Edge device layer concerns peripheral devices such as smart phones that connect to the device layer. These provide local control of the device and act as gateway between the device and the public cloud. In practice these two layers can also be combined into one device.

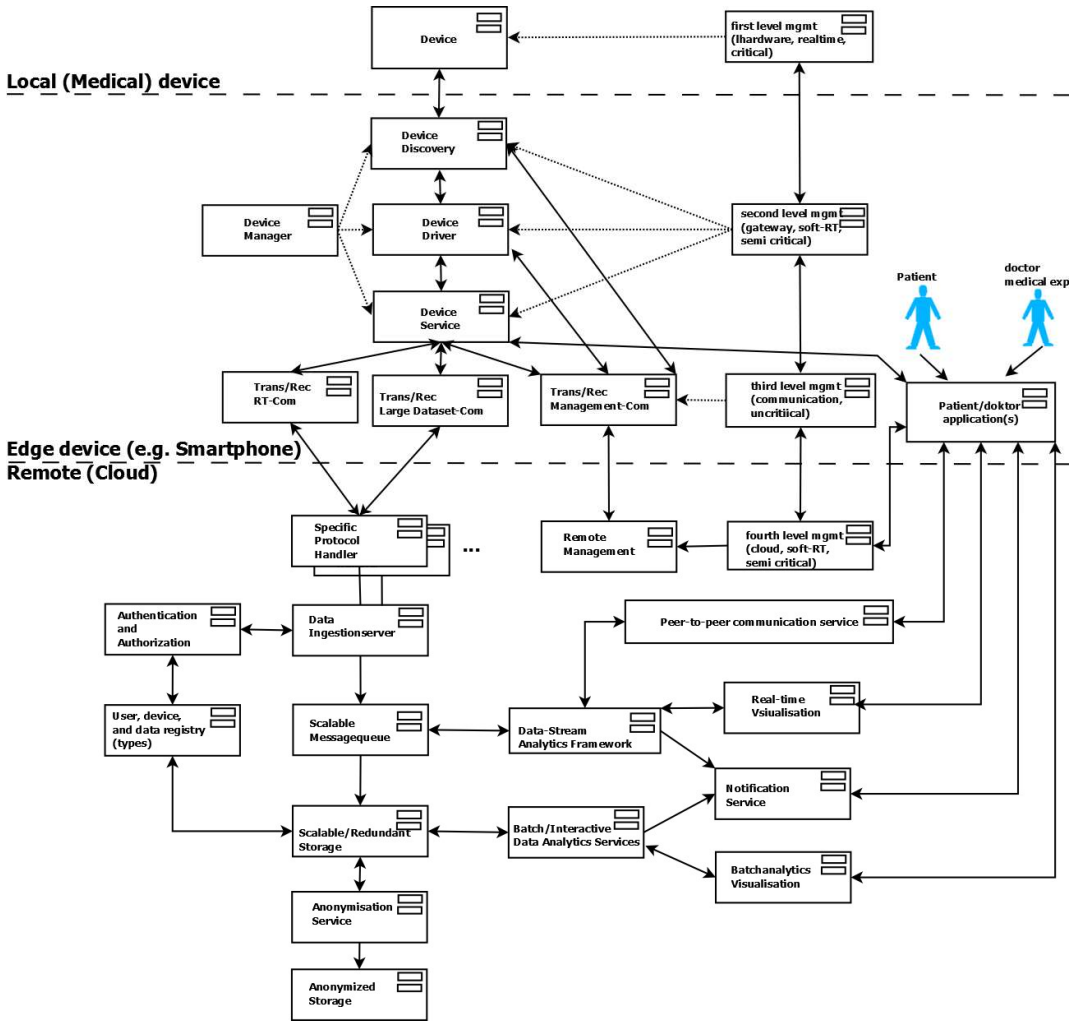


Figure 4: Functional view of Medolution as a System

The following sections provide a summary description of each component in each individual layer.

5.1.1 Local (medical) device

The local (medical) device is a hardware device including embedded software performing measurements or delivering treatments.

Device

Responsibility: actual hardware measurement or treatment unit including embedded software performing measurement or delivering treatment.

Dependencies: connects via wired or wireless communication to device discovery running on separate hardware (for example smartphone). Connects also to the first level management also available on the hardware device for critical (live threatening) management operations.

Examples: LVAD which delivers data to a telemedical center and can be managed from the outside, Simple blood pressure gages, implantable devices like ICDs, wearable's etc.

First level management
Responsibility: runs critical control management software directly on the device and might send notifications to second-level-management is necessary
(e.g. battery status critical).
Dependencies: Interacts directly with the device and exposes an API to allow control of critical device parameters from the edge device or cloud (higher-level management components). Critical management functions triggered from the cloud will only affect the device through this first level management to be blocked on any critical conflicts.
Examples: Embedded software that monitors and controls critical device parameters e.g. lvad flow control or battery management.

5.1.2 Edge Device (e. g. Smartphone)

The local edge device can perform operations affecting the medical device connected to it or might operate as a gateway to send data from the device to the cloud or vice versa. Therefore several components are necessary for fault tolerant operation.

Device discovery
Responsibility: Hardware and/or software responsible for discovery of devices offering access throughout a generic interface to connect drivers to the device. This allows usage of device through different communication technologies even device virtualization can be implemented using the discovery as a binary forwarding between machines. This would cause virtualization on the hardware level and is most likely.
Dependencies: A device the discovery is connected to (e.g. a bluetooth device like a LVAD), a Device Manager to handle the discovered devices and a Device Driver to connect the discovered Device to handling the binary data.
Examples: Discovery of devices connected via Bluetooth, RS232, TCP/IP or even high level protocols such as UPnP, or virtual devices sending training or test data.

Device driver
Responsibility: A device driver implements an API to a physical system and thus makes the devices accessible by an IT-system or a control unit. The device driver API is the low(est)-level interface in a system.
Dependencies: Has direct access to the hardware device which will be controlled by the driver and provides services for consuming data or controlling devices.
Examples: An example can be a driver which interprets the binary protocol of a hardware LVAD device to offer a generic LVAD service for consuming and controlling LVADs.

Device service
Responsibility: hosting of services on the edge device that allows functional access to a device class. The device can be read by several data sinks on the device. Sinks might

be apps using the data locally or also data transmitter/receiver making the data available to the outside.
Examples: E.g. and Heartware HVAD will be presented as a LVAD measurement service and a LVAD control service which will apply for all LVADs regardingless the vendor. This allows telemedical vendor independent operation of an LVAD.

Device Manager
Responsibility: Composition of the Device Discovery, Device Driver and Device Service. It is performing the matching between the components and orchestrates the connections.
Dependencies: Device Discovery, Device Driver and Device Service.
Examples: Bluetooth hardware LVAD is connected through the Bluetooth-Discovery to a Hardware LVAD driver offering an LVAD device service which was done by a matching procedure based on hardware address and friendly name of the device.

Secondary level management
Responsibility: Semi-critical management software running on the edge device. Second level management performs operations through first level management which has higher priority in case of conflicting management goals.
Dependencies: first level management and if existing third level management.
Examples: A management algorithm running on the smartphone adapting LVAD flow to decrease battery draining.

Third level management
Responsibility: runs semi critical management operations which are processed on the cloud. Third level management performs operations through second level management which has higher priority in case of conflicting management goals and will pass its decision to the first level management.
Dependencies: Fourth level management if existing and second level management.
Examples: A management algorithm running in the cloud optimizing LVAD controls regarding optimal flow based on big data analysis in the cloud.

Trans/Rec real-time communication
Responsibility: streaming of real time measurement data (large amount of small data packets) optimized for low latency and high availability. It is separated from large non-real time data and management data.
Dependencies: peer-to-peer communication with external system that supports same protocol and device services on the other hand to communicate to.
Examples: MQTT message broker using a non-persisting asynchronous configuration.

Trans/Rec Large dataset communication
Responsibility: Streaming of non-real time measurement data (large amount of big data packets) optimized for guaranteed transfer. It is separated from fast real time data and management data.
Dependencies: peer-to-peer communication with external systems that support the same protocol.

Examples: MQTT message broker optimized for guaranteed transfer of every data package independent from speed of the connection or size of the date. In theLVAD use case used from images transfer.

Trans/Rec Management communication
Responsibility: streaming of real time management data (large amount of small management data packages) optimized for guaranteed and fast transfer. Separated from fast real time data and large file data packages) optimized for guaranteed transfer.
Dependencies: peer-to-peer communication with external systems that support the same protocol.
Examples: MQTT message broker optimized for guaranteed and fast transfer of every data package to be able to reconfigure data transfer from the device on connection issues.

Patient/Doctor application(s)
Responsibility: A suite of applications that are used either by patients or doctors which connect to one or more Medolution cloud services to provide specific functionality. For example: showing alerts, showing real-time or batch data or setting up peer-to-peer communications services. It can also connect to a local Device Service for consuming real-time measurements or controlling devices.
Dependencies: Depend on concrete implementations of cloud services and local device connections that provide notifications, visualizations etc.
Examples: A device management dashboard application, a video conferencing application, a sensor monitor etc.

5.1.3 Remote (Cloud)

Remote management
Responsibility: Cloud hosted service to which devices connect to transmit their management data. Provides an API to retrieve this management data.
Dependencies: receives or transmits management data from or to the device.
Examples: MQTT for transmission of management data, for example battery power. REST API's can be used to retrieve data from remote management.

Fourth level management
Responsibility: Provides a cloud-hosted service that interprets management data and provides an API to perform non-critical remote control of a device.
Dependencies: Passes resulting state/configuration changes to third level management which applies these changes to the device via second level management. Retrieves management data from Remote Management.
Examples: Retrieving diagnostic information from the LVAD device and setting some alerting thresholds.

Specific protocol handler
Responsibility: Receives data from devices via a device-specific protocol and provides a device protocol-independent API towards external services that ingest this data.
Interfaces: implements the specific protocol required by a device. Streams received data to data ingestion components.
Examples: MQTT message broker for protocol, signal-R for data streaming.

Data ingestion server
Responsibility: Ingests multiple data streams corresponding to individual sensors, combines these and puts the resulting data in a message queue.
Dependencies: Connects to protocol handlers to ingest data, and forwards this data to a message queue where external services can pick up the resulting data.
Examples: Kafka

Scalable message queue
Responsibility: provides a message queue that can handle many concurrent publishers and subscribers.
Interfaces: provides API's to publish messages and subscribe to messages.
Examples: RabbitMQ, Kafka, Cassandra connector.

Scalable/Redundant storage
Responsibility: Persistent, redundant and secure storage of data.
Dependencies: None, services push or pull data from and to this database
Examples: Cassandra Database provides all of these capabilities.

Anonymisation service
Responsibility: Anonymizes data and puts this data into Anonymized storage.
Dependencies: Receives non-anonymized data from storage, and puts the anonymized data into anonymized storage.
Examples: Annonimization solution as implemented by Norima in Medolution.

Anonymized storage
Responsibility: Provides anonymized data for big data analytics purposes.
Dependencies: None.
Examples: Standard scalable databases.

Authentication and Authorization
Responsibility: Provides a centralized service that provides creation of authorization and/or authentication tokens based on user, device or data type profiles.
Dependencies: Uses profiles from User-device and data registry.
Examples: Services that implement OAut2 authentication or the SAML protocol.

User, device and data registry
Responsibility: Provides a service to perform registration of users, user-devices and data streams produced by devices. Combined with the authentication and authorization

service this allows trusted retrieval, storage and access to medical or personal data of a user.
Dependencies: None.

Peer-to-peer communication service
Responsibility: Provides a peer-to-peer connection between applications consisting out of video, text, voice or other bi-directional communications.
Dependencies: Used by applications running on edge devices.
Examples: WebRTC services.

Data-Stream analytics Framework
Responsibility: Performs real-time analytics of data that is streamed to cloud. This results in metadata (Derived measurements, alerts etc.) that are streamed to other services, like the notification service or the real-time visualization service.
Dependencies: Depends on real-time data that is streamed to cloud. This can be consumed by connecting to device data pushed to the message queue, or consumed from other services, like the peer-to-peer communication service.
Examples: Real-time analytics of video, measurements coming from medical devices. Examples: Storm, Kafka Streams

Real-time Visualization
Responsibility: Provides applications containing visualizations that show real-time data analytics results in the form of dashboards, plots, trends etc.
Dependencies: Receives data from the real-time data analytics framework.

Batch/Interactive Data Analytics Services
Responsibility: Performs non-real-time analytics of data that is stored in cloud storage. This results in metadata (Derived measurements, alerts etc.) that are streamed to the notification service and batch analytics visualization applications.
Dependencies: Retrieves data from cloud storage.
Examples: R-Applications, lambda services.

Notification service
Responsibility: Provides a service to which consumers can subscribe to receive notifications.
Dependencies: Alerts are pushed to this service by data analytics services.
Examples: email notifications, mobile messages etc. in-app notifications.

Batch Analytics visualization
Responsibility: Provides applications containing visualizations that show batch data analytics results in the form of dashboards, plots, trends etc.
Dependencies: Receives data from the real-time data analytics framework. R-Framework (Shiny for Web-Apps based on R analytics)

5.2 Best practice technology principles

This section provides an overview of best practice architectural principles when designing a Medolution-like system. Based on the decomposition provided in Figure 4, during implementation or the off-the-shelf selection of individual component technologies the following best practices are recommended:

- Design components in a stateless way so that they are not affected greatly by a server crash or relaunch and can be scaled to add redundancy and resilience. As a general framework, the twelve factor-app pattern (https://12factor.net/) provides a good reference.
- Design service API's as REST interfaces defined using the OpenAPI specification. Examples of such API's are the data ingestion API's as developed in the LVAD use case.
- Use of off-the-shelf components and frameworks such as Cassandra, Kafka, NIFI, Storm and lamba-based architectures to build big data analytics solutions as these components provide key functions such as database redundancy and security, scaleability, etc.
- Split information streams of high frequency realtime device data, device management data and large, low frequency data, to improve the reliability of individual streams.
- Implement device management as a layered system from the device layer up to the cloud management layer.

5.2.1 Dependability in Cloud deployment

In cloud infrastructures, availability is a key metric for measuring dependability. It is common to evaluate how well-designed a cloud-based system is by its ability to operate with an availability of 99.99% or more. Another important factor is of course maintainability or the measure of how fast, reproducible and easily a system can be deployed or restored to operational status after a certain failure. The following common strategies are recommended for ensuring high availability and disaster recovery in cloud-based systems. As a best practice it is adviced to select a PaaS solution that provides the following capabilies:

- Server replication across two or more availability zones, including database mirroring;
- Enabling auto-scaling to dynamically optimize performance when the application is subject to variable traffic;
- Using a managed Domain Name System (DNS) for dynamic traffic re-routing in case an outage is detected;

- Automating backup and recovery processes;
- Use monitoring and alert messages to be promptly notified of any occurring problems;
- A custom service catalog of services that can easily be composed into a working, medical device compliant solution. Compliance in this context includes compliance to standards provided in chapter 4. An overview of such required services is given in Figure 4.
- A centralized Identity Access Management solution which seamlessly integrates with all available services.
- Use of standard deployment and life cycle management technologies, like Tosca to accelerate application design and deployment.

The BDCF solution provided by Bull in this project is an example of a PaaS platform that facilitates the design and operation of a high-availability system, by providing an API to manage cloud infrastructure and set up most of the functionalities described above. More information on the BCDF platform can be found in D4.4 – Medolution Platform APIs and Specification V2. The BCDF platform is used by several demonstrators developed in the Medolution project.

5.2.2 Dependability in device-based systems

Dependability of device-based systems is an important aspect of Medolution-like systems. However, definition of dependability-related requirements and ensuring dependability requirements are met is a non-trivial exercise. There are several ways to ensure embedded system dependability. One approach is the model-based engineering (MBE) approach, which offers the advantage of starting safety analysis early in the development phase. In the following, we will briefly describe the advantages of the MBE approach for the development of heterogeneous systems and their architectures using the Architecture Analysis & Design Language (AADL).

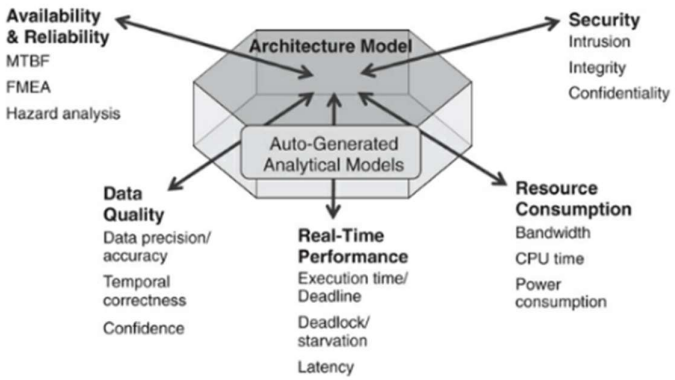


Figure 5 - Benefit of MBE with AADL (Feiler and Gluch 2012)

Figure 5 shows the benefit of model-based driven development. This approach allows us to perform dependability analyses such as fault-tree, fault impact and latency analyses during the design and planning phases and to see architectural changes automatically in these analyses. (Feiler and Gluch 2012)

AADL offers a great advantage in modelling dependability over other modelling languages such as UML and SysML. One advantage is the Error-Model-Annex which can be used to model and analyse errors, the error behaviour of components and the error propagation in the modelled system. (Feiler and Rugina 2007) This enables a fast and targeted development of safety-critical systems as required in the Medolution project.

5.2.2.1 Example of AADL

The following examples of AADL modelling in Medolution show a blueprint how to incorporate dependability analysis and specific design in critical parts of the Medolution environment.

Figure 6 shows what technology architecture in AADL looks like using the example of the LVAD use case, in particular the aortic valve cleaning cycle application. For an early dependability analysis, the architecture model does not need to be completely defined. The software components for example do not have to be modelled at a level that represents the individual threads and the hardware does not have to describe the memory or processor that will be used in the end system. It is only important that all components of the system that are relevant for the safety analysis are modelled. (for example, those for which the behaviour in the presence of an error is considered) (Feiler and Rugina 2007)

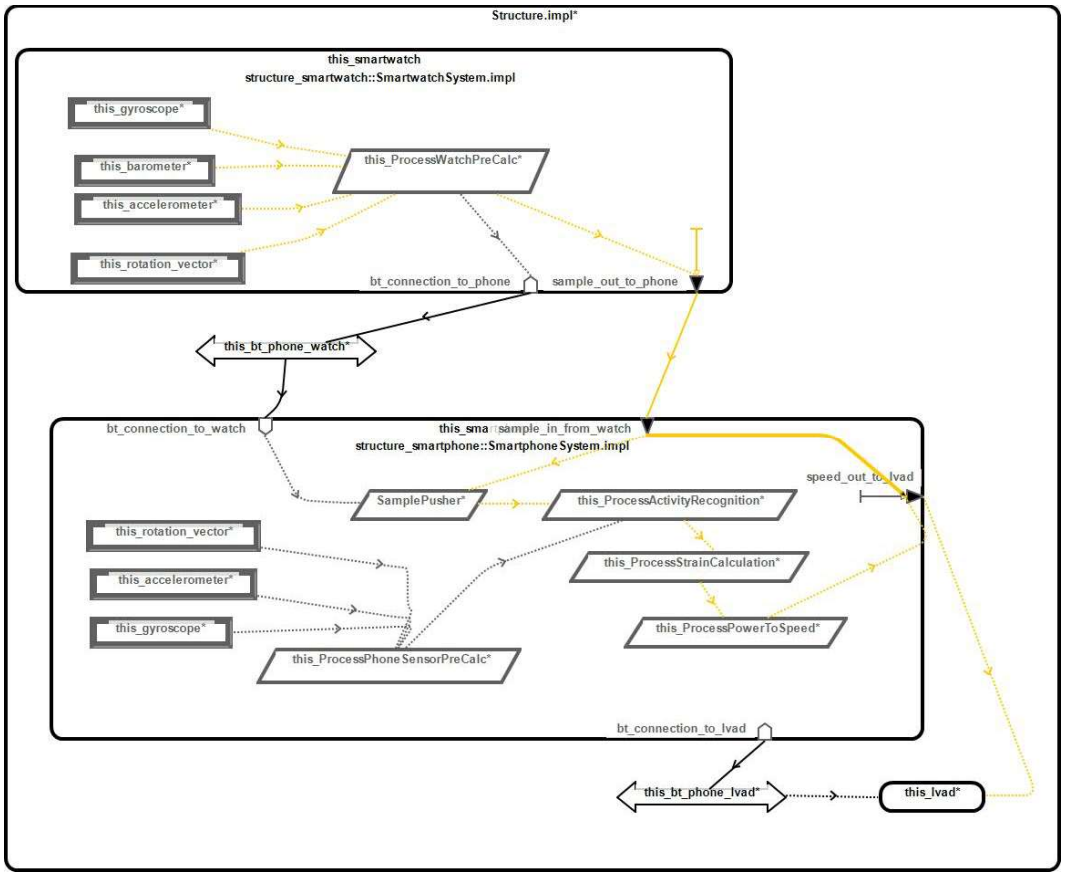
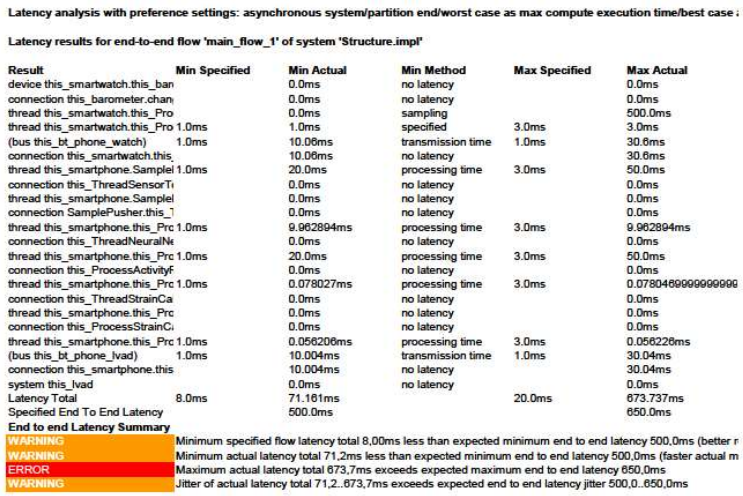


Figure 6 - AADL model of two subsystems connected via Bluetooth communication

Figure 6 illustrates the two components Smartwatch (which is shown on the left side) and Smartphone (which is shown on the right side) which are connected via a communication bus. The yellow marked line is a defined data flow that has one or more sources and is routed in the system to a sink. This has the advantage that on the one hand the exact data transfer in the system and all involved components are displayed and on the other hand we have the possibility in AADL to specify latencies either for a complete end-to-end flow as well as for individual components or connections.



Latency analysis with preference settings: asynchronous system/partition end/worst case as max compute execution time/best case :

Latency results for end-to-end flow 'main_flow_1' of system 'Structure.impl'

Result	Min Specified	Min Actual	Min Method	Max Specified	Max Actual
device this_smartwatch.this_ban		0.0ms	no latency		0.0ms
connection this_barometer.chan		0.0ms	no latency		0.0ms
thread this_smartwatch.this_Pro		0.0ms	sampling		500.0ms
thread this_smartwatch.this_Pro	1.0ms	1.0ms	specified	3.0ms	3.0ms
(bus this_bt_phone_watch)	1.0ms	10.06ms	transmission time	1.0ms	30.6ms
connection this_smartwatch.this_		10.06ms	no latency		30.6ms
thread this_smartphone.Samplel	1.0ms	20.0ms	processing time	3.0ms	50.0ms
connection this_ThreadSensorT		0.0ms	no latency		0.0ms
thread this_smartphone.Samplel		0.0ms	no latency		0.0ms
connection SamplePusher.this_T		0.0ms	no latency		0.0ms
thread this_smartphone.this_Prc	1.0ms	9.962894ms	processing time	3.0ms	9.962894ms
connection this_ThreadNeuralNe		0.0ms	no latency		0.0ms
thread this_smartphone.this_Prc	1.0ms	20.0ms	processing time	3.0ms	50.0ms
connection this_ProcessActivityF		0.0ms	no latency		0.0ms
thread this_smartphone.this_Prc	1.0ms	0.078027ms	processing time	3.0ms	0.0780469999999999
connection this_ThreadStrainCa		0.0ms	no latency		0.0ms
thread this_smartphone.this_Prc		0.0ms	no latency		0.0ms
connection this_ProcessStrainCa		0.0ms	no latency		0.0ms
thread this_smartphone.this_Prc	1.0ms	0.056206ms	processing time	3.0ms	0.056226ms
(bus this_bt_phone_lvad)	1.0ms	10.004ms	transmission time	1.0ms	30.04ms
connection this_smartphone.this		10.004ms	no latency		30.04ms
system this_lvad		0.0ms	no latency		0.0ms
Latency Total	8.0ms	71.161ms		20.0ms	673.737ms
Specified End To End Latency		500.0ms			650.0ms

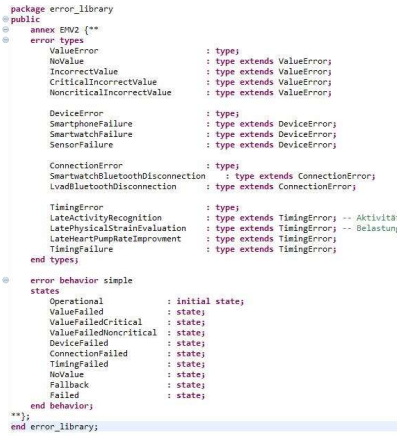
End to end Latency Summary

WARNING	Minimum specified flow latency total 8,00ms less than expected minimum end to end latency 500,0ms (better r
WARNING	Minimum actual latency total 71,2ms less than expected minimum end to end latency 500,0ms (faster actual m
ERROR	Maximum actual latency total 673,7ms exceeds expected maximum end to end latency 650,0ms
WARNING	Jitter of actual latency total 71,2..673,7ms exceeds expected end to end latency jitter 500,0..650,0ms

Figure 7 - Part of an end-to-end latency analysis in AADL

Figure 7 shows how latency analysis was automatically generated based on the dependability information provided to the model (flow and latency information). Based on the specified minimum and maximum latency information that the data transfer requires according to requirements (min. 500ms, max. 650ms), and the available time information such as transfer time and processing time of individual architectural components, we recognize in an early development phase that our current system architecture does not meet the required requirements with regard to maximum end-to-end latency. (max end-to-end latency of 673ms)

The early detection of faults and the automatic analysis/documentation of dependability aspects enable the development of safety-critical systems. In addition to the dataflow information, AADL also enables the definition of user-defined errors.



```
package error_library
public
    annex EMV2 {**
    error types
        ValueError                  : type;
        NoValue                     : type extends ValueError;
        IncorrectValue              : type extends ValueError;
        CriticalIncorrectValue      : type extends ValueError;
        NoncriticalIncorrectValue   : type extends ValueError;

        DeviceError                 : type;
        SmartphoneFailure           : type extends DeviceError;
        SmartwatchFailure           : type extends DeviceError;
        SensorFailure               : type extends DeviceError;

        ConnectionError             : type;
        SmartwatchBluetoothDisconnection    : type extends ConnectionError;
        LvadBluetoothDisconnection  : type extends ConnectionError;

        TimingError                 : type;
        LateActivityRecognition     : type extends TimingError; -- Aktivitä
        LatePhysicalStrainEvaluation    : type extends TimingError; -- Belastun
        LateHeartPumpRateImprovment     : type extends TimingError;
        TimingFailure               : type extends TimingError;
    end types;

    error behavior simple
    states
        Operational             : initial state;
        ValueFailed             : state;
        ValueFailedCritical     : state;
        ValueFailedNoncritical  : state;
        DeviceFailed            : state;
        ConnectionFailed        : state;
        TimingFailed            : state;
        NoValue                 : state;
        Fallback                : state;
        Failed                  : state;
    end behavior;
**};
end error_library;
```

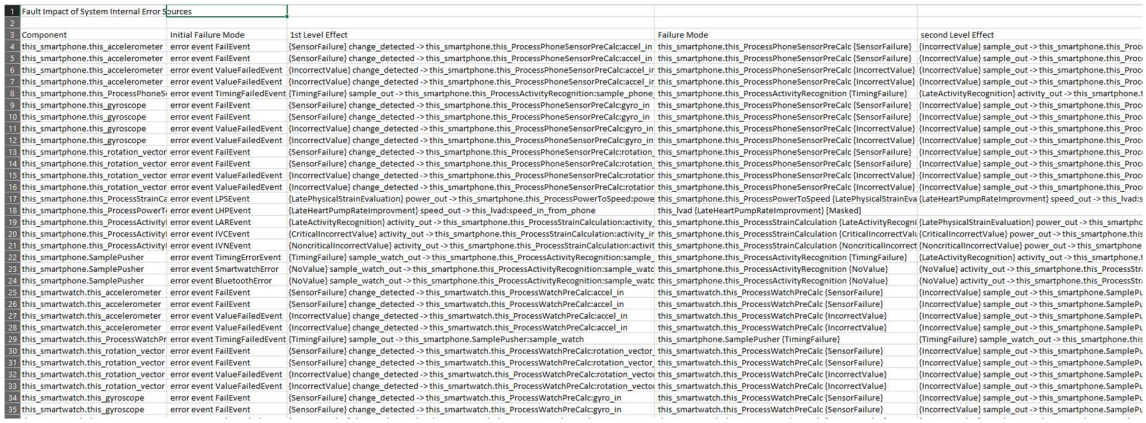
Figure 8 - Error definition and behaviour model in AADL

```
annex EMV2 {**
    use types error_library;
    use behavior error_library::simple;
    error propagations
        rotation_vector_in :in propagation{DeviceError,IncorrectValue};
        gyro_in :in propagation{DeviceError,IncorrectValue};
        accel_in :in propagation{DeviceError,IncorrectValue};
        sample_out :out propagation{TimingFailure,IncorrectValue};
    flows
        f0 :error path rotation_vector_in{DeviceError,IncorrectValue} -> sample_out{IncorrectValue};
        f1 :error path gyro_in{DeviceError,IncorrectValue} -> sample_out{IncorrectValue};
        f2 :error path accel_in{DeviceError,IncorrectValue} -> sample_out{IncorrectValue};
        f3 :error source sample_out{TimingError} when TimingFailed;
    end propagations;

    component error behavior
        events
            TimingFailedEvent: error event;
        transitions
            t0: Operational -[rotation_vector_in{DeviceError,IncorrectValue} or gyro_in{DeviceError,IncorrectValu
                accel_in{DeviceError,IncorrectValue}
            ]-> ValueFailed;
            t1: Operational -[TimingFailedEvent]-> TimingFailed;
        propagations
            p0: ValueFailed -[]-> sample_out{IncorrectValue};
            p1: TimingFailed -[]-> sample_out{TimingFailure};
    end component;
    properties
        -- TODO needed?
        EMV2::OccurrenceDistribution => [ProbabilityValue => 0.1; Distribution => Poisson;] applies to TimingFail
        --EMV2::OccurrenceDistribution -> [ProbabilityValue -> 0.1; Distribution -> Poisson;] applies to f3;
**};
```

Figure 9 - Example of an error definition of a software component in AADL

Figure 8 shows how user-defined errors are created in AADL and that they can be subdivided even more precisely. (type extends) Furthermore, it is possible to define states in which a system or individual components can be located. (behaviour states) Using this information in combination with the defined system structure and flow information, it is possible to visualize and analyse the occurrence of errors and their propagation in a modelled system. Figure 9 shows an example of how to extend an AADL component regarding error information. The information about the occurrence and propagation of errors in the system can be used for automated fault impact analyses.



Fault Impact of System Internal Error Sources

Component	Initial Failure Mode	1st Level Effect	Failure Mode	second Level Effect
this_smartphone.this_accelerometer	error event FailEvent	{SensorFailure} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:accel_in	this_smartphone.this_ProcessPhoneSensorPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_accelerometer	error event FailEvent	{SensorFailure} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:accel_in	this_smartphone.this_ProcessPhoneSensorPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_accelerometer	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:accel_ir	this_smartphone.this_ProcessPhoneSensorPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_accelerometer	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:accel_ir	this_smartphone.this_ProcessPhoneSensorPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_ProcessPhoneSe	error event TimingFailedEvent	{TimingFailure} sample_out -> this_smartphone.this_ProcessActivityRecognition:sample_phone	this_smartphone.this_ProcessActivityRecognition {TimingFailure}	{LateActivityRecognition} activity_out -> this_smartphone.t
this_smartphone.this_gyroscope	error event FailEvent	{SensorFailure} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:gyro_in	this_smartphone.this_ProcessPhoneSensorPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_gyroscope	error event FailEvent	{SensorFailure} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:gyro_in	this_smartphone.this_ProcessPhoneSensorPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_gyroscope	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:gyro_in	this_smartphone.this_ProcessPhoneSensorPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_gyroscope	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:gyro_in	this_smartphone.this_ProcessPhoneSensorPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_rotation_vector	error event FailEvent	{SensorFailure} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:rotation_	this_smartphone.this_ProcessPhoneSensorPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_rotation_vector	error event FailEvent	{SensorFailure} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:rotation_	this_smartphone.this_ProcessPhoneSensorPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_rotation_vector	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:rotatior	this_smartphone.this_ProcessPhoneSensorPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_rotation_vector	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartphone.this_ProcessPhoneSensorPreCalc:rotatior	this_smartphone.this_ProcessPhoneSensorPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.this_Proc
this_smartphone.this_ProcessStrainCa	error event LPSEvent	{LatePhysicalStrainEvaluation} power_out -> this_smartphone.this_ProcessPowerToSpeed:powe	this_smartphone.this_ProcessPowerToSpeed {LatePhysicalStrainEva	{LateHeartPumpRateImprovment} speed_out -> this_lvad:s
this_smartphone.this_ProcessPowerT	error event LHPEvent	{LateHeartPumpRateImprovment} speed_out -> this_lvad:speed_in_from_phone	this_lvad {LateHeartPumpRateImprovment} {Masked}	
this_smartphone.this_ProcessActivity	error event LAREvent	{LateActivityRecognition} activity_out -> this_smartphone.this_ProcessStrainCalculation:activity_	this_smartphone.this_ProcessStrainCalculation {LateActivityRecogni	{LatePhysicalStrainEvaluation} power_out -> this_smartphc
this_smartphone.this_ProcessActivity	error event IVCEvent	{CriticalIncorrectValue} activity_out -> this_smartphone.this_ProcessStrainCalculation:activity_ir	this_smartphone.this_ProcessStrainCalculation {CriticalIncorrectVali	{CriticalIncorrectValue} power_out -> this_smartphone.this
this_smartphone.this_ProcessActivity	error event IVNEvent	{NoncriticalIncorrectValue} activity_out -> this_smartphone.this_ProcessStrainCalculation:activit	this_smartphone.this_ProcessStrainCalculation {NoncriticalIncorrect	{NoncriticalIncorrectValue} power_out -> this_smartphone
this_smartphone.SamplePusher	error event TimingErrorEvent	{TimingFailure} sample_watch_out -> this_smartphone.this_ProcessActivityRecognition:sample_	this_smartphone.this_ProcessActivityRecognition {TimingFailure}	{LateActivityRecognition} activity_out -> this_smartphone.t
this_smartphone.SamplePusher	error event SmartwatchError	{NoValue} sample_watch_out -> this_smartphone.this_ProcessActivityRecognition:sample_watc	this_smartphone.this_ProcessActivityRecognition {NoValue}	{NoValue} activity_out -> this_smartphone.this_ProcessStr
this_smartphone.SamplePusher	error event BluetoothError	{NoValue} sample_watch_out -> this_smartphone.this_ProcessActivityRecognition:sample_watc	this_smartphone.this_ProcessActivityRecognition {NoValue}	{NoValue} activity_out -> this_smartphone.this_ProcessStr
this_smartwatch.this_accelerometer	error event FailEvent	{SensorFailure} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:accel_in	this_smartwatch.this_ProcessWatchPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_accelerometer	error event FailEvent	{SensorFailure} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:accel_in	this_smartwatch.this_ProcessWatchPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_accelerometer	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:accel_in	this_smartwatch.this_ProcessWatchPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_accelerometer	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:accel_in	this_smartwatch.this_ProcessWatchPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_ProcessWatchPr	error event TimingFailedEvent	{TimingFailure} sample_out -> this_smartphone.SamplePusher:sample_watch	this_smartphone.SamplePusher {TimingFailure}	{TimingFailure} sample_watch_out -> this_smartphone.this
this_smartwatch.this_rotation_vector	error event FailEvent	{SensorFailure} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:rotation_vector_	this_smartwatch.this_ProcessWatchPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_rotation_vector	error event FailEvent	{SensorFailure} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:rotation_vector_	this_smartwatch.this_ProcessWatchPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_rotation_vector	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:rotation_vecto	this_smartwatch.this_ProcessWatchPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_rotation_vector	error event ValueFailedEvent	{IncorrectValue} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:rotation_vecto	this_smartwatch.this_ProcessWatchPreCalc {IncorrectValue}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_gyroscope	error event FailEvent	{SensorFailure} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:gyro_in	this_smartwatch.this_ProcessWatchPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.SamplePu
this_smartwatch.this_gyroscope	error event FailEvent	{SensorFailure} change_detected -> this_smartwatch.this_ProcessWatchPreCalc:gyro_in	this_smartwatch.this_ProcessWatchPreCalc {SensorFailure}	{IncorrectValue} sample_out -> this_smartphone.SamplePu

Figure 10 - Fault impact analysis

Figure 10 shows the automatically generated document, which lists the individual components in the system and their error states with associated error propagation. This document can be used to determine the individual components that are all affected in the event of a specific error. With additional stochastic information such as the

probability of errors occurring and their distribution, a fault tree analysis can be performed, as shown in Figure 11.

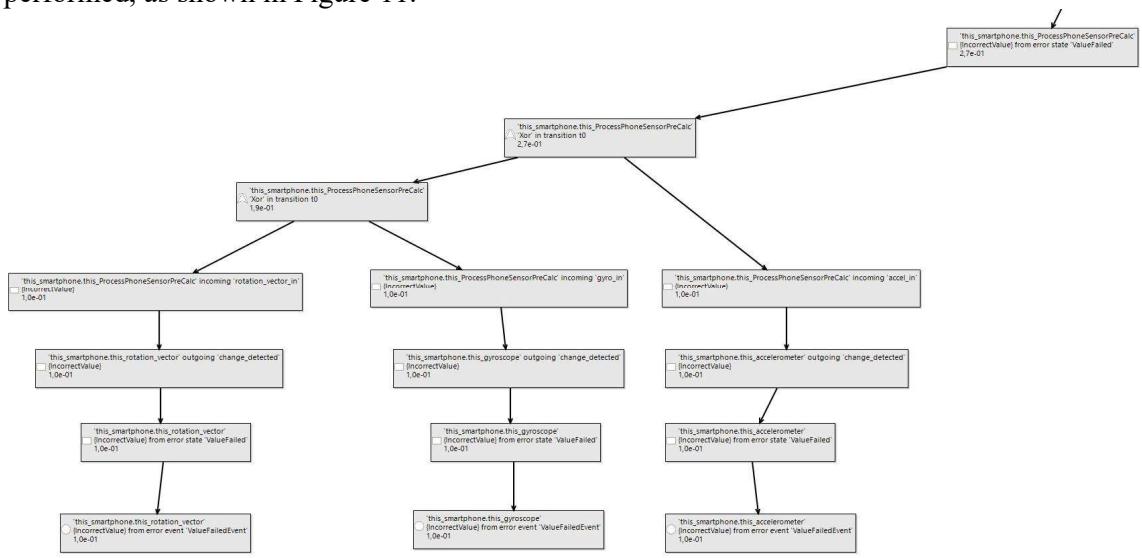


Figure 11 - Fault tree analysis

The fault tree analysis is used to determine the causes of faults and their probability for a specific fault event. Fault tree analysis is a top-down method that is often used for risk analysis. This means that you specify an event that should not occur (the TOP event) and work your way down in a tree structure from the top to the roots (the origin of the TOP event). The fault tree analysis is a very frequently used method for detailed fault analysis. It is suitable for finding event combinations (roots in the tree) that lead to an error event (TOP event). The aim is to connect events by logical AND / OR / XOR connections and to represent these different events in various levels. The higher-level event is always the result of the underlying error. The fault tree analysis is used both to analyse causes and to plan or improve the system architecture (Feiler and Delange 2017). All these opportunities for modelling and analysis guarantee the development of safety-critical systems. By extending system architecture with suitable fault tolerance mechanisms, specific errors in the system are masked and not propagated or suitable system configurations are carried out to repair these errors. This is documented by the analysis and provides improved analysis results in an early development phase if the MBE approach is used with AADL.

5.3 Conclusions of this chapter

This chapter provides a common architectural view that covers all Medolution use cases. Besides this, an overview of architectural best practices is provided for both cloud-based and device-based systems.

6 Conclusions

There is a clear need for Medolution-like systems. This document provides ideas on how to develop business models for Medolution like systems while also providing an overview of current legislations that apply to systems handling personal and medical data. Given the current legislations and business expectations, a significant progress in business and social acceptability of Medolution-like systems should be made.

From a technology perspective, this document provides a common architectural view including derived architectural best practices that covers the individual Medolution demonstrators. It is thus concluded that although the demonstrators target different use cases, their technology background can be mapped to a single and consistent technological view. This supports the idea that a Medolution-like system will in practice be a combination of heterogenous, independently developed, systems. This is in line with the observations made in the Business Architecture presented in this document. As dependability is a clear architectural driver for Medolution-like systems, this document presents an overview of architectural best practices when incorporating dependability into a Medolution-like system.

Medolution demonstrators show technological implementation examples of this technology architecture.

In order to get more detailed information on specific best-practice aspects presented in this document the reader is recommented to read the information sources provide in the table below.

Best practice gategories	**Detailed information source**
General best practices	This document, section 5.2
Cloud dependability	This document, section 5.2.1, **D4.4** – Medolution Platform APIs and Specification V2
Device based dependability and interoperability	This document section 5.2.2, **D3.2-3.4-3.5** – Guideline for development of Big Dependable System components to enable short time to market interoperability between medical devices and BDSs + Software release
Business aspects	This document, chapter 3
Information system aspects	This document, chapter 4, **D1.1** – State of the Art Analysis. Super cloud project: https://supercloud-project.eu/

7 Publication bibliography

Feiler, Peter; Delange, Julien (2017): Automated Fault Tree Analysis from AADL Models. In *ACM SIGAda Ada Letters* 36 (2), pp. 39–46. DOI: 10.1145/3092893.3092900.

Feiler, Peter; Rugina, Ana (2007): Dependability Modeling with the Architecture Analysis & Design Language (AADL). In *Dependability Modeling with the Architecture Analysis & Design Language (AADL):*

Feiler, Peter H.; Gluch, David P. (2012): Model-Based Engineering with AADL. An Introduction to the SAE Architecture Analysis & Design Language: Addison-Wesley Professional.