Expertise
and insight
for the future

Tarvainen Mika

# Software System Architecture Development in a Regulated Environment

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

15 November 2018

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Mika Tarvainen<br>Software System Architecture Development in a Regulated Environment<br>62 pages + 4 appendices<br>15 November 2018 |
| Degree | Master of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Health Technology |
| Instructors | Mika Järvenpää, Senior Project Manager<br>Sakari Lukkarinen, Senior Lecturer |

The main goal of this thesis was a research of requirements and methods for software system architecture development and description in the field of medical and in vitro diagnostic (IVD) medical device manufacturing. The work was conducted for a company as a part of a product development project during transition periods of European medical and IVD medical device regulations, which came into force in May 2017. Due to the new laws and other changes in the regulated environment, to the objectives of the study were included reviewing of product and software development standard operation procedures (SOPs) of the company and recommending updates to those and researching the new regulations for an analysis of responsibilities for manufacturers who supply components to IVD medical device systems.

The focus in the first part of the theory section of the thesis is on the new laws and on harmonised standards related to the laws. Standard ISO 13485 and IEC 62304 are highlighted, as they are the most relevant ones for product and software development. The second part of the theory concentrates on architecture development generally and particularly for medical device software. The method for the architecture description was applying of commonly known architecture view model and unified modelling language (UML) diagrams.

The analysis of the company SOPs produced several suggestions for improvements. The question of responsibilities for manufacturers supplying components to IVD systems was answered based on literature review of the new regulations. The views to the architecture were documented using a modified version of a commonly known view documentation template. The architectural design emerged to a model driven and event driven software system architecture utilizing architectural patterns suitable for embedded systems. The type of system is a typical materialization of a medical device.

The outcomes of the study were evaluated qualitatively. Method for the evaluation was interviewing. The interviewees were from the company and represented various stakeholders of the project and the thesis. All of them found the level of significance of the study high and considered the thesis to be well structured and of good quality.

| | |
|---|---|
| Keywords | medical device, in vitro diagnostic, IEC 62304, software system architecture, pattern, model driven |

## Contents

## List of Abbreviations

| | |
|---|---|
| ADL | Architecture Description Language. A method for describing system and software architectures in a systematic fashion. |
| ATAM | Architecture tradeoff analysis method. A formal process for evaluating software architectures. |
| AUTOSAR | AUTomotive Open System ARchitecture. Standardized software architecture for automotive industry. |
| CFR | Code of Federal Regulations (CFR). U.S. law. |
| COMM | Communication. COMM modules or ports are physical entities capable of communicating using some specific communication standard and/or protocol. |
| DHF | Design History File. Medical device design and development documentation as required by FDA. |
| EDA | Event Driven Architecture. An architectural style for systems where events trigger actions. |
| EU | European Union. |
| FDA | Food and Drug Administration of United States. Regulates medical devices in U.S. |
| GDPR | General Data Protection Regulation. Regulation EU 2016/679 on personal data and free movement of the personal data. |
| GUI | Graphical User Interface. |
| IDE | Integrated Development Environment. A software combining software source code editor with other tools, such as compiler, linker and debugger. |

| | |
|---|---|
| IEC | International Electrotechnical Commission. An organization developing international standards for electrical and related technologies. |
| IEEE | Institute of Electrical and Electronics Engineers. An organization developing international standards for example for telecommunications and information technology. |
| IO | Input Output. Information transfer in computer system. Often relates to binary input/output pins in microprocessor systems. |
| ISO | International Organization for Standardization. An organization developing worldwide standards for different industries. |
| IVD | In Vitro Diagnostic. 'In glass' (Latin) diagnostics, procedures performed outside a human body using samples originated from the human body. |
| IVDMD | In Vitro Diagnostic Medical Device. An IVD medical device is regulated by EU 2017/746 instead of regulation EU 2017/745 for medical devices. |
| IVDR | In Vitro Diagnostic medical device Regulation. EU regulation 2017/746. |
| LOC | Level of Concern. Risk estimation scale in FDA's guidance document *"Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices"*. |
| MD | Medical Device. Regulated by EU 2017/745. |
| MDA | Model Driven Architecture. A design method where a system is visually modelled prior implementation of it according to the model. |
| MDCG | Medical Device Coordination Group. A committee of experts providing advice and assistance for European Commission on MDR and IVDR implementation. |
| MDEG | Medical Devices Expert Group. Experts from medical device industry, public health sector, notified bodies and standardisation organisations. The |

| | |
|---|---|
| | group is responsible of creating MEDDEV guidance for repealed MD and IVDMD directives. |
| MDR | Medical Device Regulation. EU regulation 2017/745. |
| NA | Not applicable. |
| NB | Notified Body. A designated body performing assessment on conformity to a regulation. |
| OS | Operating System. The basic software in computers that provides essential services for applications running in top of it. |
| PEMS | Programmable Electrical Medical System. IEC 60601-1 definition for medical device combining hardware and software. |
| PUI | Physical User Interface. |
| RUO | Research Use Only. RUO products are intended for scientific applications without any medical purpose. |
| SDD | Software Design Description. Documentation of a software as defined in standard IEEE 1016. |
| SIS | Supplementary Information Sheet. Info-page for an international standard from FDAs recognized consensus standards database. |
| SOP | Standard operating procedure. Instructions for carrying out some operation in a uniform manner. |
| SOUP | Software of Unknown Provenance or “off-the-shelf software” as in IEC 62304. |
| UDI | Unique Device Identification. Unique identifier code of a medical device consisting of device and product identifiers. |
| UI | User Interface. |

| | |
|---|---|
| UML | Unified Modelling Language. A visual tool for modelling systems. Mostly used in software engineering. |
| V&V | Verification & Validation. Testing activities performed during and after a product development project. Verification is done against specifications and validation against intended use. |
| QSR | Quality System Regulation. FDA's equivalent to European quality systems management standard ISO 13485. |

# 1 Introduction

The regulated environment in which medical device manufacturers operate has gone through some changes recently. The former European medical device and In Vitro Diagnostic medical device directives were replaced with regulations EU 2017/745 and EU 2017/746 in May 2017 [1], [2]. In addition to the changed legislations, a new version of the quality system management (for medical devices) standard ISO 13485 was released in 2016. This standard was already harmonised with the directives and is the main guidance for ensuring that medical device life cycle processes are applied like intended by the regulations [3]. Harmonising means that the standard is listed as one of many standards constituting an overall guidance for manufacturers how to comply with a regulation in question. Conforming to the harmonised standards is highly recommended [4].

IEC 62304 is another harmonised standard entirely for software life cycle processes. This is the standard where it is stated that software requirements must be transformed into an architecture if the medical device software safety classification is at certain level. [5, cp. 5.3]. Standard operating procedures (SOPs) for developing and maintaining medical device software are conveniently build by following the IEC 62304 chapter by chapter. Manufacturers exporting products also to markets outside Europe need to be familiar with the local regulations of the markets in question. For example, in United States the party responsible of medical device regulations is the Food and Drug Administration (FDA). Similarly like IEC 62304, FDA requires software architecture documentation if the level of concern of the software is at certain level [6, table 3].

The aim of this thesis is to investigate how to design and develop a software system architecture of a medical or In Vitro Diagnostic (IVD) medical device in the manner that the outcome is compliant with the related medical device regulations. The objectives of the study include analyzing the requirements of medical device software architecture, exploring related theory and methods and presenting an example of an implementation loosely coupled to a real-life product development project of a company.

First, the product development SOPs of the company are analyzed and recommendations for updating them to meet the requirements of changed regulations in

Europe and of all other related regulations and standards are made. The focus is on software development.

In addition, a difference between requirements for manufactures of components used in medical devices and manufacturers of medical devices are explained in the new context of updated regulations and standards. The researching of this subject is especially requested by the company.

Finally, an example of a software system architecture is presented in such a generic level of detail that a reader not familiar with the business area of the company can constitute a perspicuous understanding of the subject. The study is closed with a qualitative evaluation of the outcome. This is performed by conducting a series of stakeholder interviews.

# 2 Background

In this chapter, the domain of regulated medical device manufacturing is explained. The narrative is written in a period when the industry was reacting to a set of prominent changes in the medical device regulations. The focus is promptly directed to software development and furthermore to software architecture development in the field in question. The last part of the chapter is a research of methods for designing and describing software architectures generally and applicably to medical device software development. The topics of design and development are approached from a viewpoint of embedded systems. Such systems combine mechanics, electronics and software and are often part of safety critical devices [7, p. 3].

## 2.1 The Regulated Environment

The regulated environment addressed in this thesis stands for medical and In Vitro Diagnostic medical device manufacturing under legislation of European Union (EU) considering also the requirements in United States given by U.S. Food and Drug Administration (FDA).

### 2.1.1 Medical Device Legislations and Harmonised Standards

The European legislation means specifically regulations EU 2017/745 for medical devices and EU 2017/746 for IVD medical devices. These regulations overrode former directives 93/42/EC and 98/79/EC in May 2017 [1], [2]. In practice, the directives are still in effect during the transition periods of the regulations. The transition periods will end in 2020 for medical devices and in 2022 for IVD medical devices. Products that are certified prior 25th May 2017 can continue to be placed on European market until the certificates of them expire. If products are certified in accordance of the directives during the transition period, then the certificates of them will expire 27th May 2024 latest [1, Article 120], [2, Article 110].

A medical device is by definition and for example an *"instrument"* or *"software"* intended *"by the manufacturer"* to be used *"for human beings"* for a *"specific medical purpose"*. The purposes include diagnosing, monitoring and similar, but also *"providing information by means of in vitro examination of specimens derived from the human body"* [1, Article 2, (1)]. The regulation EU 2017/745, in which the definition is given, states that it shall

apply to the defined type of products but not apply to In Vitro Diagnostic medical devices which are regulated by the regulation EU 2017/746 instead [1, Article 1, 6, (a)].

The federal government of United States recognizes the IVD products also:

> *"In vitro diagnostic products are those reagents, instruments, and systems intended for use in the diagnosis of disease or other conditions, including a determination of the state of health, in order to cure, mitigate, treat, or prevent disease or its sequelae. Such products are intended for use in the collection, preparation, and examination of specimens taken from the human body."* [8]

Products matching with the definition above are considered as medical devices and are regulated in United States by the U.S. Food and Drug Administration (FDA) [9]. The specific U.S. law for medical devices is the code of federal regulations (CFR) Title 21 – Food and Drugs [10].

EU provides lists of standards related to different directives and regulations like medical device and in vitro diagnostic medical device regulations [4]. The standards in these lists are said to be harmonised with the regulations. The definition of a harmonised standard explains the importance of such:

> *"A harmonised standard is a European standard developed by a recognised European Standards Organisation: CEN, CENELEC, or ETSI. It is created following a request from the European Commission to one of these organisations. Manufacturers, other economic operators, or conformity assessment bodies can use harmonised standards to demonstrate that products, services, or processes comply with relevant EU legislation."* [4]

Indeed, both the medical device regulation (MDR) and the IVD medical device regulation (IVDR) presume that conformity with the list of *"the related harmonised standards"* published in *"the Official Journal of the European Union"* provide conformity with the regulations [1, Article 8], [2, Article 8]. In the middle of 2018, EU has not yet published the lists for the regulations. Instead, the latest updates to the lists of harmonised standards for MD and IVDMD directives were published in official journal C389 in the end of 2017 [11], [12].

FDA has a recognized consensus standards program that recognises international standards. The program has a database from which a supplementary information sheet (SIS) can be obtained for any standard. If a standard is recognized, then the SIS of it contains information about publications and guidance documentation that are relevant to the standard [13]. The guidance documentation is FDA's interpretation of the U.S. law and following the guidance provides a comprehensive compliancy with FDA's regulations or CFR [14].

EU has guidance for repealed MD and IVDMD directives in a form of so-called MEDDEVs, created by Medical Devices Expert Group (MDEG). MDR and IVDR regulations introduced a new group called Medical Device Coordination Group (MDCG). The tasks of the group include assisting and advising the European commission on MDR and IVDR implementation. In addition, the MDCG creates guidance documents for the new regulations. The EU's guidance is not legally binding, but following of it is highly expected. [15]

When comparing former MD and IVDMD directives and the guidance on those to the new MDR and IVDR, it is evident that many topics of the MEDDEVs are now addressed by the new legislation. For example MDR Annex XIV *"Clinical Evaluation and Post-market Clinical Follow up"* is aligned with MEDDEV 2.7/1 revision 4 *"Clinical Evaluation: a Guide for Manufacturers an Notified Bodies Under Directives 93/42/EEC and 90/385/EEC"*. Two more correlations are between MDR chapter VII *"Post-Market Surveillance, Vigilance and Market Surveillance"* and MEDDEV 2.12-1 rev 8 *"Guidelines on a Medical Devices Vigilance System"* and between MDR chapter IV "Notified Bodies" and MEDDEV 2.10-2 Rev. 1 *"Designation and Monitoring of Notified Bodies within the Framework of EC Directives on Medical Devices"* [1], [15]. A notified body (NB) is a designated conformity assessment body for the regulations [1, Article 2]. The conformity assessment procedure must be undertaken before a medical device can be placed on the European market [1, Article 52]. The identification number of a notified body responsible for a conformity assessment of a medical device is marked below a CE marking of the device [1, Article 20].

The first guidance document from the Medical device coordination is MDCG 2018-1 on Unique Device Identification (UDI) system. UDI system is a new requirement in the regulations compared to the former directives [15]. Manufacturers are required to place an UDI code to every individual medical device and to register the codes to a UDI database, which is part of the new Eudamed database of the European Commission.

The UDI requirement is expected to improve traceability of devices especially in case of safety related incidents [1, Articles 25 - 30]. In mid-2018, the Eudamed database is not yet established. FDA has an UDI requirement and a database also. FDA's Global UDI Database has been in use since 2013 [16].

There are several other changes in the MDR, as compared with the directives that it has repealed. The main theme is ensuring safety of medical devices by emphasising life cycle management and requiring clinical evidence. The scope of the MDR is broader than the directive, for example, online services and some products without intended medical intention are included. Furthermore, notified bodies are under more strict control, manufacturers are required to name a regulatory person responsible for MDR compliance and post-market activities and responsibilities for the manufacturers are required in detail. There are also changes in the risk classification rules, which may cause some products to be re-classified. [17]

The most notable change in the IVDR is the new risk classification system that causes all IVD products to be re-classified. The new classification system is rule based, instead of the list-based system in the directive. The four risk classes A, B, C and D are in an increasing order of risk to individual and/or to public health. The classes are summarized in the Figure 1.

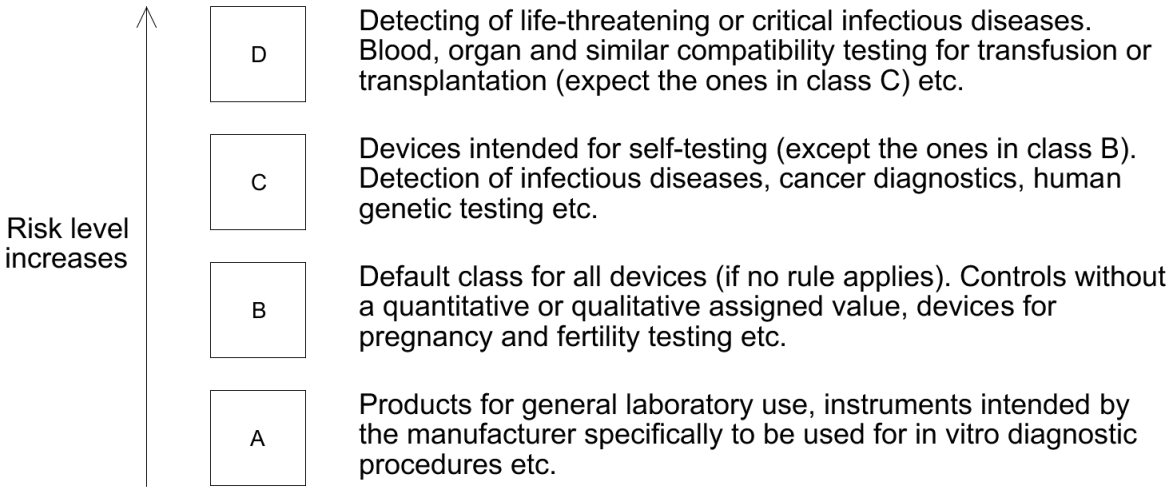


Figure 1. IVDR risk classes. [2, Annex VIII]

Conformity assessment by a NB is required for products in classes B, C and D. In addition to the new risk classification system, the IVDR contains plenty of same changes to manufacturers and NBs as the MDR. Clinical evidence, performance evaluation, performance studies and post-market performance follow up are in a focus. The clinical evidence is expected to prove *"scientific validity, analytic validity and clinical performance*". [18]

### 2.1.2 Quality Management Systems

One of the essential standards both in EU's MD and IVDMD lists of harmonised standards is ISO 13485 or "*Medical devices -- Quality management systems -- Requirements for regulatory purposes*" [11], [12]. Relatively new version of it was released in 2016. Among other requirements, ISO 13485 requires manufacturers to document design and development procedures [3, cp. 7.3.1]. These are often called SOPs or standard operating procedures. The standard also requires that "design and development":

> *"is planned, controlled and documented; inputs and outputs for it are reviewed, maintained and documented; is reviewed, verified and validated; transferring it to manufacturing is done using documented procedures; changes to it are controlled and approved"*. [3, cp. 7.3.1 - 7.3.9 ]

The requirements listed in the chapter 7.3 of the standard cover all design and development activities for a whole life cycle of a medical device. In addition, it is required in the chapter that a design and development file shall be maintained for storing all the documents and records generated by cause of the requirements listed above. The purpose of the file is to prove the conformity to the requirements and maintain history of all changes in a design [3, cp. 7.3.10 ].

FDA has a quality system regulation similar to European quality management systems standard ISO 13485, commonly known as quality system regulation (QSR) or part 820 of CFR title 21. Requirements for design and development are listed in the subpart C (§820.30) as "Design Controls" and they are quite identical to the requirements in ISO 13485. The name for the design and development file by FDA is a design history file (DHF). [10]

The changes in the ISO 13485:2016 compared to ISO 13485:2003 are conveniently available in a separate comparison table in annexes of the standard. Software development is not directly affected; word *"software"* is only mentioned in changes that are related to validation of software used in medical device production and in the quality management system itself. The above-mentioned requirement for design and development file is added as a new sub-clause to the 2016 version of the standard. As are requirements for complaint-handling processes, validation of processes for sterilization and sterile barrier systems, competence of people involved in product quality and processes and measures for training the people in question. [3, Table A.1]

### 2.1.3 IEC 62304 and Related Standards

IEC 62304 *"Medical device software – Software life cycle processes"* is a harmonised standard focusing entirely on software. In the very beginning of it, the standard recommends that meeting customer and regulatory requirements for a software can be proven by using a quality management system compliant with ISO 13485. These requirements are inputs for the five software life cycle processes required by the IEC 62304 for medical device manufacturers to establish; development, maintenance, risk management, configuration management and problem resolution. The core of the development and the maintenance processes is presented in the Figure 2. [5]

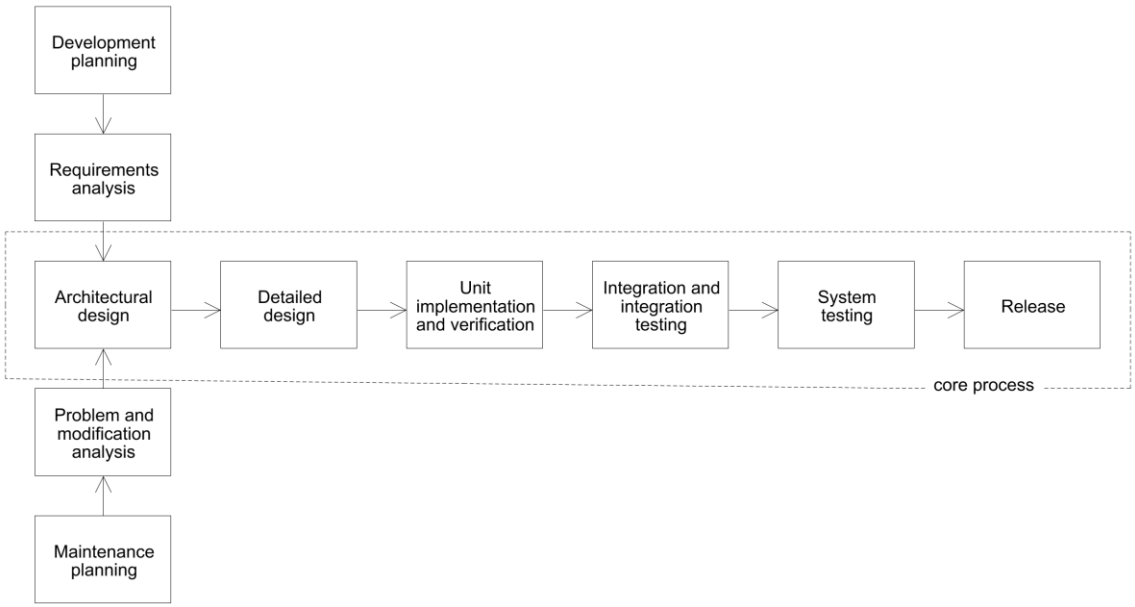


Figure 2. The core of development and maintenance processes of IEC 62304. [5, Fig. 1 & 2]

As can be seen in the Figure 2, the development activities both in the development and in the maintenance processes are the same. The core process takes an input either from system requirements or from change/problem management. Software architecture may be affected first, then the design and implementation may be altered and finally the changes are verified and the whole system is validated. It is notably, that software testing, verification and validation are part of the development and maintenance processes. The core process of them may be a part of a product development V-model as described in IEC 62304 and illustrated in the Figure 3. [5]

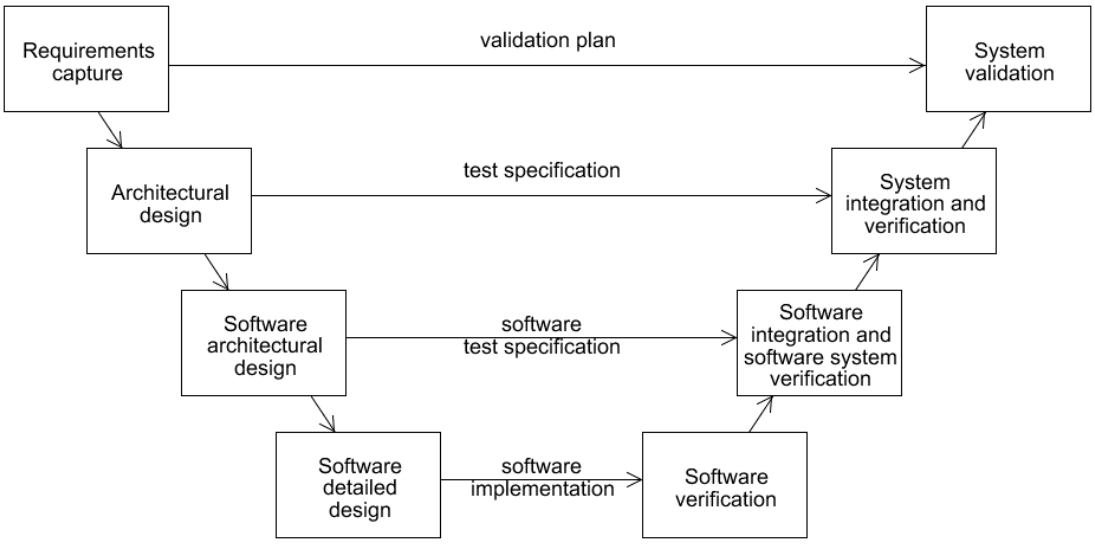


Figure 3. V-model. [5, Figure C.2]

The design stream on the left side of the V-model provides inputs for the testing stream on the right. The use of the V-model is not mandatory according to IEC 62304. However, it is a reference to another standard IEC 60601-1 *"Medical electrical equipment Part 1: General requirements for basic safety and essential performance"*, in which the model is presented and where a development life cycle is required [5, cp. C.4.3]. The V-model can be understood as an extension of classical Waterfall model, which is presented in the Figure 4. In the V-model, the testing activities of the Waterfall-model are bend up to the right stream to emphasis relations to the development activities.

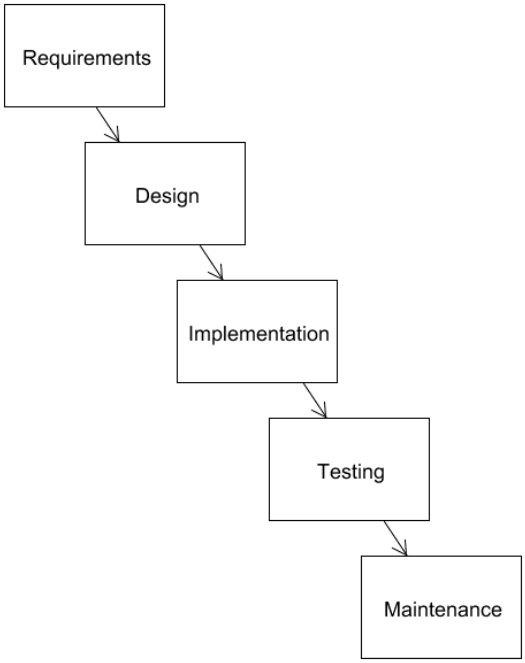


Figure 4. Waterfall-model. [19, Figure 1]

In Waterfall-model, each activity creates a set of requirements for the next activity. A review of requirements can be organized before an activity is allowed to start. The model implements a top-down ideology for system development [19, p. 62, Fig. 1]. A disadvantage of the Waterfall model is that every activity needs to be completed before next one can be started. Any problem in the downstream may cause re-design in the previous activities. Methods such as Agile address this problem by allowing the development activities iterate in short cycles. Requirements for the activities can be re-visited during the development and maintenance life cycle. Activities in iterative processes produce multiple outputs and are active simultaneously [20, cp. 3.2].

In addition to the “once-through” Waterfall method, IEC 62304 recognizes also iterative strategies. Technically, they are called “incremental” and “evolutionary” in the standard. In both of them, there occur multiple development cycles and they may distribute interim software. An evolutionary model does not require defining of all of the software requirements in the beginning of the development. Moreover, it definitely aims at delivering interim software. IEC 62304 permits use of any life cycle model. However, the standard underlines that before releasing *“any medical device software”* all outputs of all five life cycle processes required by the standard should be observed and validated for consistency with each other. [5, cp. B.1.1]

IEC 62304 life cycle process “configuration management” manages software items, and their versions and history, created during development or maintenance processes and included in a medical device software. It also contains change control activity for ensuring that only approved changes are implemented. The change control activity records change requests and approvals of those and it maintains traceability from the requests to the actual changes in the software and to software verification of the changes. Another goal of the activity is to prevent unintentional changes in addition to unapproved ones [5, cp. B.8]. One source of change request inputs for the configuration management process is a “problem resolution” process [5, cp. 9.2]. The process is alternatively called “defect-tracking” and the purpose of it is to document, analyse and resolve any issue, problem or non-conformance found during a life cycle of a medical device software. Evaluation of problems in the problem resolution process is essential when estimating the risks related to safety of a medical device [5, cp. B.9].

Along with quality management system, IEC 62304 requires that the software life cycle processes are executed within a risk management system and precisely in compliance with standard ISO 14971 *“Medical devices – Application of risk management to medical devices”*. This is addressed by the “risk management” process, which is the fifth and last IEC 62304 life cycle process introduced in this chapter. The basis for the software risk management is an IEC 62304 software safety classification, which describes the severity of harm that a software failure can create to a patient or to other people. The requirements in the standard for activities and for level of documentation differ between different safety classes. The classification process is further explained in the chapter 2.2 of this document [5, cp. 4]. For each software system in a medical device, excluding the ones with lowest risk, the manufacturers are required to identify those software modules or items that could be involved in hazardous situations analyzed in system risk analysis. Causes including defects, incorrect functionality and failures of third party software need to be analyzed and documented. Risk control measures shall be defined, analyzed, implemented, verified and documented. Documentation shall include traceability from hazardous situations to software items, risk control measures and verification of those [5, cp. 7]. FDA recognizes the risk management standard version ISO 14971:2007 completely [21]. The harmonised version in EU is ISO 14971:2012 [11].

IEC 62304 is greatly influenced by industry independent standard ISO 12207 *“Systems and software engineering – Software life cycle processes”*. The main difference is that the IEC 62304 focuses more on risk management and safety. The amount of processes in IEC 62304 is also reduced by choosing the ones that are useful for medical device

development [5, cp. C.6]. Other sources of inspiration for IEC 62304 are ISO 90003 *"Software engineering -- Guidelines for the application of ISO 9001 to computer software"* and IEC 61508 *"Functional safety of electrical/electronic/programmable electronic safety-related systems"* (especially *"Part 3: Software requirements"*) [5, cp. C.1]. ISO 90003 references standard ISO 9001 *"Quality management systems – Requirements"*, which is the base of ISO 13485 [3, cp. 0.4]. The positions of the mentioned non-harmonised standards in IEC 62304 context are clarified in the Figure 5.

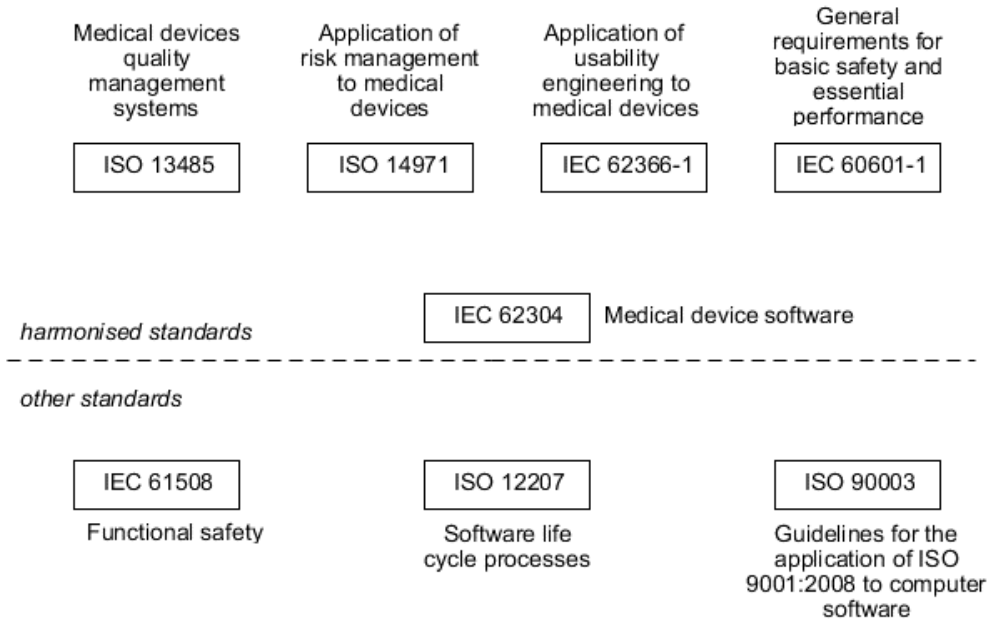


Figure 5. IEC 62304 –centric viewpoint to the most relevant medical device software standards. [5, Figure C.1]

Among the most relevant standards for medical device software development is the IEC 60601-1 *"Medical electrical equipment Part 1: General requirements for basic safety and essential performance"*. IEC 62304 addresses the requirements in IEC 60601-1 for programmable electrical medical system (PEMS). The foremost idea in IEC 60601-1 is that PEMS devices are so complex that merely testing the final product does not guarantee the safety of it. Instead, the development of a PEMS has to follow systematic processes in addition to testing [22, clause 14]. IEC 60601-1 also challenges medical device manufacturers' ability to develop user interfaces that do not contribute to safety related problems. A usability engineering process is required for achieving *"reasonable usability"* [22, clause 12].

Usability engineering is expected to be conducted according to a harmonised standard IEC 62366-1 *"Medical devices - Part 1: Application of usability engineering to medical devices"*. The target of the usability engineering is to achieve safe usability of medical devices by minimizing use errors that could lead to hazardous situations. The process is performed by analyzing how, by whom, and in what kind of environment a medical device is used and what kind of risks there could be for patient and user safety when the device is used in the specified conditions [23]. IEC 62304 demands that software requirements need to contain user interface requirements implemented by software, for example related to *"areas needing concentrated human attention"* and then simply refers to IEC 62366-1 and to a standard IEC 60601-1-6 "*Medical electrical equipment - Part 1-6: General requirements for basic safety and essential performance - Collateral standard: Usability"* [5, cp. 5.2.2]. It is also mentioned in IEC 62304 that software system testing should cover usability [5, cp. B.5.7]. FDA recognizes usability standard IEC 62366-1 completely and the equivalent FDA guidance is *"Applying Human Factors and Usability Engineering to Medical Devices - Guidance for Industry and Food and Drug Administration Staff"* [24].

There are more references to other standards in IEC 62304 than is illustrated in the Figure 5. For example, system requirements for *"software that is itself a medical device"* are defined in standard IEC 82304-1:2016 "*Health software -- Part 1: General requirements for product safety"* [5, cp. 1.2]. It is also highlighted in the IEC 62304 that the safety standard IEC 60601-1 does not apply to IVD medical devices. Instead, standard IEC 61010-2 *"Safety requirements for electrical equipment for measurement, control and laboratory use – Part 2-101: Particular requirements for in vitro diagnostic (IVD) medical equipment"* does. Furthermore, the risk management clause in the general part of the IEC 61010-2, which is IEC 61010-1, is not aligned with IEC 14971. As a conclusion, IEC 62304 declares that *"if laboratory equipment is used as IVD equipment…"*, then the *"application of ISO 14971 is required for risk management"*, and if software is involved, then *"IEC 62304 must be taken into account"* [5, cp. C.5].

The first edition of IEC 62304 is over ten years old. Edition 1.1 of 2015 amended requirements for legacy software, meaning *"software, where the software design is prior to the existence of the current version"* [5, cp. Introduction]. ISO published a corrective update to IEC 62304 in November 2017 and is currently developing edition two with a working name *"IEC/DIS 62304 Health software -- Software life cycle processes"*. A new version of the risk management standard ISO 14971 for medical devices is also under development by ISO [25].

### 2.1.4 Cybersecurity and Data Privacy

According to IEC 62304, software requirements specification should include security requirements related to *"compromise of sensitive information, authentication, authorization, audit trail, communication integrity and system security/malware protection"* [5, cp. 5.2.2]. However, the standard does not address the topic further. FDAs SIS page for IEC 62304 includes two cybersecurity related guides, thus implying that FDA's guidance for cybersecurity is ahead of IEC 62304 [26]. FDA expects medical device manufacturers to set up certain cybersecurity controls for ensuring safety and security. These controls include identifying of threats and analysing of the risks involved. The cybersecurity documentation related to the controls is also required to be attached to any premarket submission submitted to FDA [27].

Europe has acknowledged the importance of information security even though the medical device software standard addresses the topic so sparsely. ENISA, the EU's agency for network and information security, has published *"Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures"* guidance document in 2017 for IoT experts, software developers and manufacturers, among others. The focus of the publication is on reliability and interoperability of IoT systems as well as on data privacy and information security. ENISA presents a high-level reference architecture or a model, which is used for clarifying concepts such as asset definitions and threat/attack identifications [28].

Information security is also one of the topics listed in MDR that have to be considered when developing a *"state of the art"* software [1, cp. 17.2]. Manufacturers are supposed to specify *"minimum requirements"* for hardware and IT networks for protecting them against unauthorised access and for ensuring information security in general [1, cp. 17.4]. MDR refers also data privacy and data protection directive 95/46/EC when setting requirements for clinical investigations [1, cp. 17.4]. The data protection directive was repealed by general data protection regulation (GDPR) in 2016. The main principles in the GDPR include lawfulness, purpose limitation, data minimisation, accuracy, confidentiality and accountability [29].

## 2.2 Software System Architecture Development in the Regulated Environment

Software is part of computer-based systems in conjunction with hardware and including documentation [30, p. 6]. The definition by Ian Somerville extends the meaning of software from being merely executable programs to a broader concept of software systems.

The standard IEC 62304 provides a more unambiguous definition for a software system: *"integrated collection of software items organized to accomplish a specific function or set of functions"* where software item is *"any identifiable part of a computer program"*. Hardware and software are combined in the standard in a definition of a system: *"integrated composite consisting of one or more of the processes, hardware, software, facilities, and people, that provides a capability to satisfy a stated need or objective"*. [5, cp. 3]

The objectives of a system consist of requirements from various stakeholders such as users, customers, legislators and business owners, as well as from some non-obvious internal parties such as system integrators and verification and validation department (V&V). In standard ISO 13485, the medical device manufacturers are obliged to determine and document these requirements [3, cp. 7.2.1].

IEC 62304 states that software requirements must be transformed into an architecture and the architecture shall contain description of interfaces between different parts of internal and external software and hardware involved if the medical device software safety classification for the software system in question is class B or C [5, cp. 5.3]. Altogether, there are three software safety classes in the standard: A, B and C. The classes are in ascending order of safety risk for patients, users or public health. Assigning the IEC 62304 software safety class for a software system is elaborated in the Figure 6 [5, cp. 4.3].

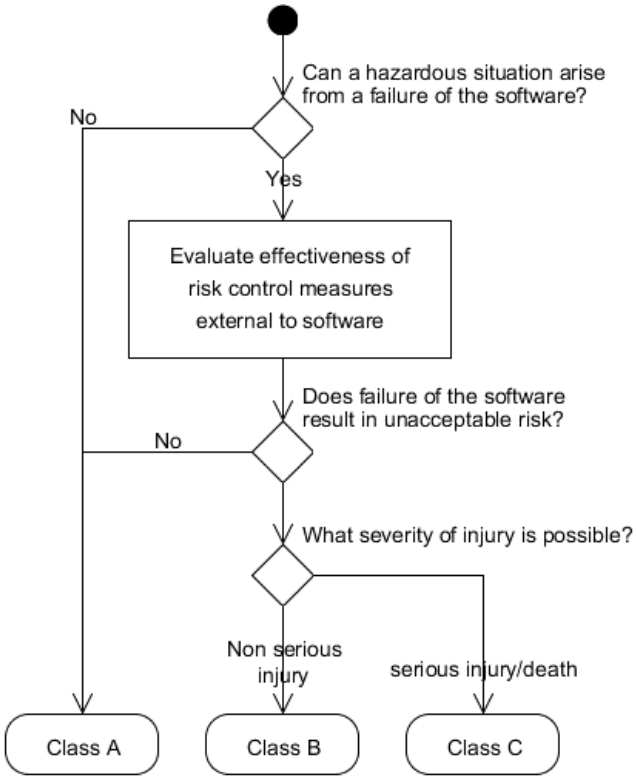


Figure 6. IEC62304 Software safety classification. [5, Figure 3]

Commonly, a risk calculation includes both severity and probability. However, the software safety classification flow chart in the Figure 6 does not take into account the probability of a software failure. The reason is that IEC 62304 considers that the probability for a software failure has to be always one (1). In addition, if risk mitigations include software, then the probability for a failure of that software has to be, again, one [5, B.4.3]. Thus the software safety class for a medical device software system is class A if the software does not cause a hazardous situation or if the hazardous situation can be mitigated with risk control measures in a system external to the software in question. Otherwise, the class is B or C, depending on the severity of an injury that the software failure may cause [5, cp. 4.3]. IEC 62304 does not require software architecture documentation for a class A software [5, cp. 5.3].

Similarly, as IEC 62304, FDA may or may not require software architecture documentation. The FDA's equivalent for the software safety class is a level of concern (LOC), explained in a FDA's guidance document *"Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices".* The document is referenced by the FDAs SIS page for IEC 62304 [26]. Level of concern can be minor, moderate or

major. FDA does not require architecture documentation for a software with minor level of concern [6, Tables 1 and 2]. However, immediately when a software device is an accessory to a medical device having at least moderate level of concern, then the LOC of the accessory software device is increased and an architecture documentation of it is required [6, Tables 2 and 3].

IEC 62304 divides the architecture requirement into six sub-requirements:

- *"Transform software requirements into an ARCHITECTURE"*
- *"Develop an ARCHITECTURE for the interfaces of SOFTWARE ITEMS"*
- *"Specify functional and performance requirements of SOUP item"*
- *"Specify SYSTEM hardware and software required by SOUP item"*
- *"Identify segregation necessary for RISK CONTROL"*
- *"Verify software ARCHITECTURE"*

These sub-requirements can be understood in the way that an architecture documentation should list different software modules and describe any interface between them. The requirement includes third party software systems or software of unknown provenance (SOUP), not demanding their architectures revealed, but their functionality and operating environment described. [5, cp. 5.3]

FDAs recommendations for an architecture design chart in a high level are quite similar than in IEC 62304: *"relationships among the major functional units in the Software Device"* [6, p. 13]. In addition, FDA recommends to add more detailed information, like state diagrams of the software functional units, if the level of concern is moderate or major [6, table 3], which is a bit confusing statement, because the architecture design chart is not required for minor level of concern software [6, table 3]. Nevertheless, IEC 62304 has similar requirement for class C software. It requires interface description of such a detailed level that the software *"units"* and interfaces between them can be implemented based on that documentation [5, cp. 5.4.3].

## 2.3 Software Architecture Concepts

The following sub-chapters introduce some basic concepts of system and software architectures. Views and viewpoints are explained as described in standard ISO 42010.

The rest of the concepts and architectural patterns are selected from those that support designing of embedded systems.

### 2.3.1 ISO 42010 – Viewing Architectures

Standard ISO 42010, *"Systems and software engineering — Architecture description"*, presents a comprehensive conceptual model for architecture descriptions not concentrating only to design and development but also to maintaining the architectures by re-visiting the descriptions during the life cycles of the architectures. The essence of the model is that stakeholders of a system have concerns about the system when they are looking at it from a certain point of view. These concerns are addressed by an architectural view to the architecture description. Some examples of the concerns are functionality, behavior, privacy and compliance to regulations. The relation of viewpoints and views is explained with two illuminating metaphors: a viewpoint is to a view as a programming language is to a program or as a legend is to a map. The standard references commonly known architecture frameworks implementing the concept such as Kruchten's 4+1 view model, which is illustrated in Figure 7. [31]

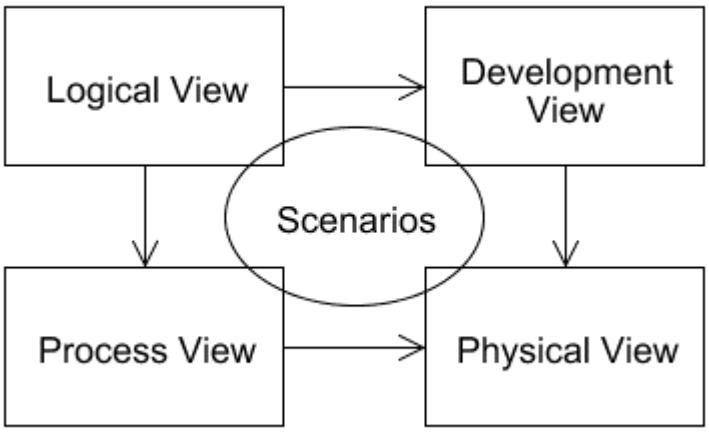


Figure 7. Kruchten's 4+1 view model. [32]

The views in the Kruchten's 4+1 model are different perspectives to a software architecture. The logical view describes end-user functionality and it is often in a form of object-oriented decomposition or class diagrams. The process view presents the task/thread/process architecture of a system and indicates on which thread the objects are processed. The development view addresses the development environment and the organization of the software in it. The main diagram of the development view is a module

decomposition structure describing the said organization. The modules use *"Layered-pattern"* by utilizing a hierarchy of layers with strict interfaces in boundaries. The physical view is a description of the computational hardware where the software is executed. The fifth view, the scenarios, is a set of use cases describing some of the most important requirements for the software as sequences of interactions between objects and processes. Similarly as ISO 42010, Philippe Kruchten declares that the different views in the model enable addressing concerns of different stakeholders separately. [32]

Kruchten mentions that the importance of different perspectives to software architectures is noticed also in Software Engineering Institute by Paul Clements, one of the authors of widely referenced book "Software Architecture in Practice" [32, p. 42]. The book presents concepts of software architecture and a set of architectural tactics how to achieve the quality attributes required by the stakeholders of a system the architecture describes. Tactics are used in practice as a part of an architectural pattern, *"a package of design decisions that is found repeatedly in practice"*. In addition, *"patterns package tactics"*, as the book explains the relation of tactics and patterns [20, cp. 13].

### 2.3.2 Tiers and Layers

The three-tier architecture is an extensively referred pattern in the undeniably vast multitude of different architectural patterns available. It is presented in the Figure 8 below.

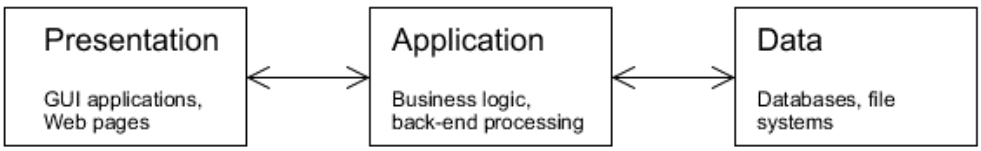


Figure 8. Three-tier architecture.

In three-tier architecture, software is divided to presentation, application and data tiers. The pattern especially enables scalability. For example, the amount of internet browsers accessing an application in a Web server can scale up easily. The data for the application may be hosted on another computing system separate from the one running the application [30, pp. 493 - 494]. Scalability is not the only quality attribute or concern to which the three-tier pattern and generally multi-tier patterns are applicable. These

patterns are said to be allocation patterns when the software processes are distributed to multiple processing hardware systems [32, p. 235]. Allocation patterns separate parts of systems from each other and can be used for segregation of software items to different software safety classes. Actually, IEC 62304 requires that medical device manufacturers identify any segregation between software items that may affect safety [5, cp. 5.3.5]. Three-tier pattern is not to be confused with equally famous three-layer patterns like the three-layer architecture of robotics, which is illustrated in the Figure 9.

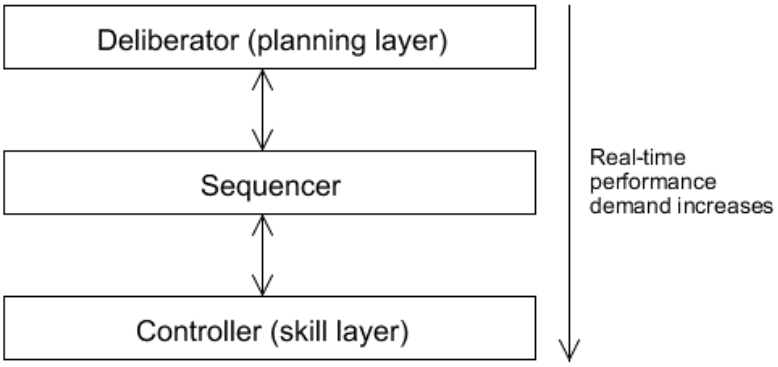


Figure 9. Three-layer architecture of robotics.

In three-layer architecture of robotics, the functions of a software system are separated by decreasing order of required real time performance to controller, sequencer and planning layers. The controller or the *"skill"* layer performs hard real-time control loops and *"primitive behaviors"* like wall-following. The sequencer executes conditional sequencing of the primitive behaviors. The planning layer or the "deliberator" layer produces plans for the sequencer and performs time-consuming computations like machine vision algorithms. The layers are allocated in different processes of a multi-tasking environment, in which the controller layer has the highest priority to access processor time. [33]

Figure 10 demonstrates another implementation of the three-layer pattern used in automotive open systems architecture AUTOSAR. The three main layers in AUTOSAR are application, runtime environment and basic software. The purpose of the layers is to separate near-hardware routines to basic software and bind hardware independent applications to it via the runtime environment [34]. The separation also correlates with

safety. Non-safety related applications such as multimedia applications are isolated from the ones controlling hardware and safety related functions [35, p. 40].

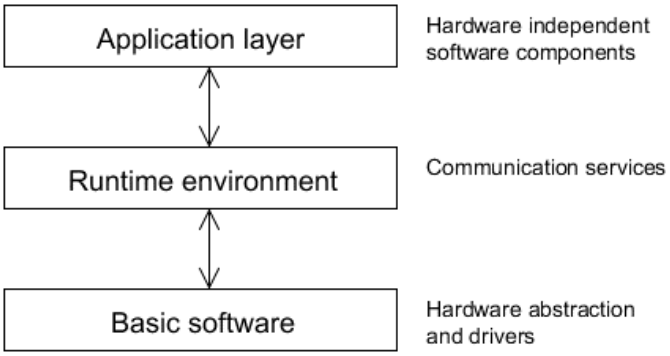


Figure 10. Three-layer AUTOSAR architecture.

AUTOSAR and three-layer architecture of robotics are applications of layered pattern. The principal rule in the pattern is that a layer is allowed to use only the next lower layer. Some exceptions to the rule can be allowed but only towards lower layers. The main quality attributes supported by the layered pattern are portability and modifiability. Porting of an application to another hardware platform can be possible by replacing only the lowest layer of the layer stack with an alternative implementation, assuming that the interface provided by the lowest layer and required by the next higher layer is well defined. Different layers or software modules can be developed independently, which elevates modifiability in systems. [20, p. 206]

### 2.3.3 Kernels and Concurrency

A kernel pattern is again another, albeit very essential, layered pattern. Kernel of an operating system (OS) provides core functionality of the OS including task scheduling and inter-process communication. Other parts of the OS and applications are layered on top of it. Kernels can be found in large operating systems and in smaller systems where the kernel is called a microkernel. The difference between patterns using a kernel or a microkernel is presented in the Figure 11. [7, p. 189]

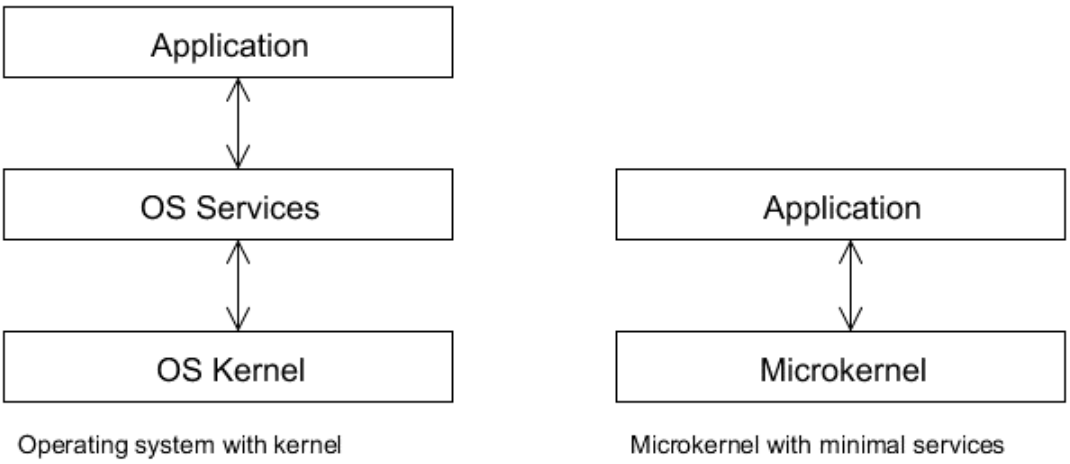


Figure 11. The difference between kernel and microkernel patterns.

Operating systems used in servers and personal computers have layered architecture where the kernel is in the bottom. Layers upwards the kernel contain services such as file systems and user management. Embedded systems often employ microkernels without additional layers or unnecessary OS services between an application and the microkernel. [7, pp. 189 - 190]

Task scheduling is the mechanism in kernels that enable concurrency in systems. A task architecture (the process view of the Kruchten's 4+1 model) describes how concurrent tasks in a system are organized, what their execution priorities are and what kind of communication scheme they are using [7, cp. 13]. Well-known communication schemes include, for instance, synchronous and asynchronous messaging patterns. In asynchronous messaging, a sender does not wait for reply, whereas in synchronous messaging the scheme is command-response based [7, cp. 11.5]. Complex control patterns may utilize both asynchronous and synchronous messaging between tasks. An example of such a pattern is the hierarchical control architectural pattern, which is presented in the Figure 12.

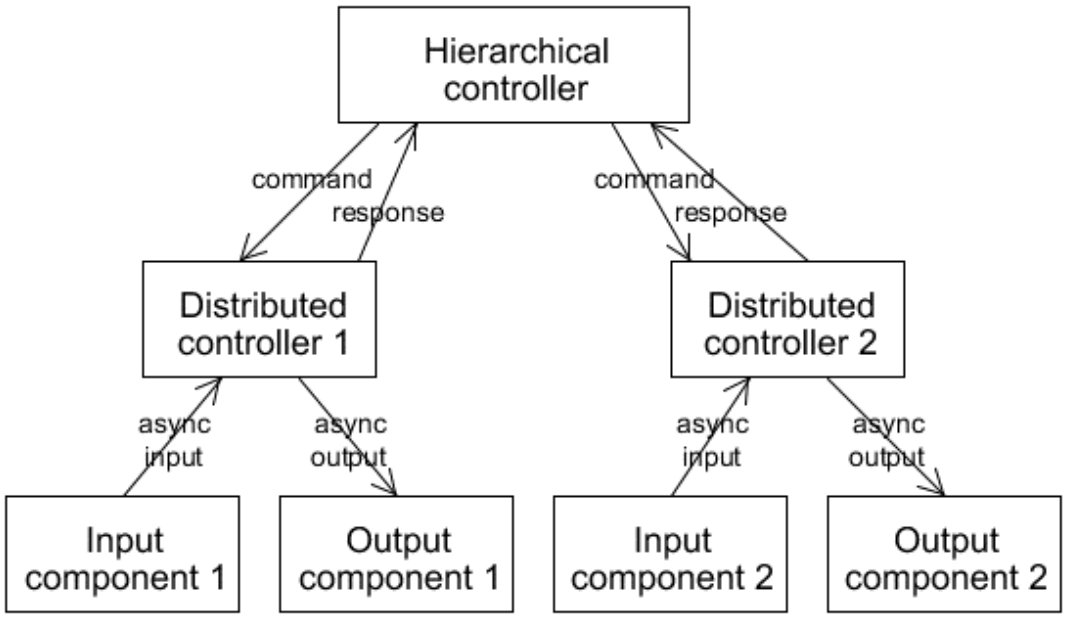


Figure 12. Hierarchical control architecture [7, Figure 11.8].

The control components of the hierarchical control architecture run state machines in their own thread of execution. Highest level of control is given to a hierarchical controller component, which assigns jobs for distributed controllers using command-response scheme. Distributed controllers receive asynchronous data from input components and set output states of output components. Slightly similar control architecture, a distributed collaborative control pattern, lacks the hierarchical controller. Instead, the distributed controllers communicate freely with each other's using event based messaging and the overall system control is in synchronization with external environment from which the events may also be generated [7, cp. 11.3].

### 2.3.4 Events and State Machines

Systems that react to events implement event driven architecture (EDA). Events are transferred asynchronously in a messaging backbone of a system. The receivers may or may not be interested in the events. If are, then the events trigger processing [36]. An example of an event processing is a state transition in a state machine: *""when event α occurs in state A, if condition C is true at the time, the system transfers to state B"*. The quotation is from famous article *"Statecharts: a visual formalism for complex systems"* by David Harel. The concept of hierarchical state machines presented in the article enabled describing behavior of state machines from different levels of complexity or

abstraction. A system can be viewed by looking only at the main states (super states) of it. If details of a functionality of a system are under interest, then the sub states of the system can be *"zoomed"* in. The zooming of modularity of super states and sub states is extended also to state transitions. Harel explains the concept with different views to a simple state machine structure, similar to the ones in the Figure 13 [37].

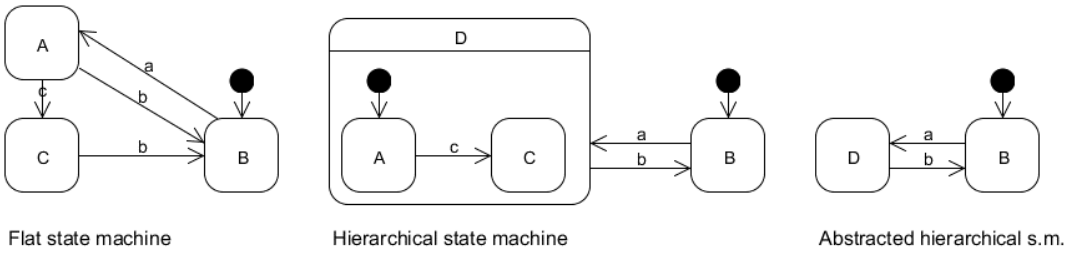


Figure 13. Difference between flat and hierarchical state machines [37, Fig. 1-6]

The *"flat"* state machine in the Figure 13 above has an initial state *"B"*. Event *"a"* triggers transition to a state *"A"*. When the machine is in the state *"A"*, then an event *"c"* creates a transition to a state *"C"*. Return back to the state *"B"* is triggered by an event *"b"* from both the states *"A"* and *"C"*. A functionally identical hierarchical version of the machine includes a super state *"D"*, which encapsulates the states *"A"* and *"C"*. The difference is that when the state *"D"* is excited by the cause of the event *"b"*, it is done whether the current sub state of the state *"D"* is state *"A"* or *"C"*. The third description presents a higher-level view to the functionality, where the insides of the state *"D"* are abstracted away. The behavior is still the same: the event *"b"* in the state *"D"* creates a transition back to the state *"B"* [37]. The state machine descriptions in the example are a form of model driven architecture (MDA). Practicing MDA means that systems are modelled prior implementing them according to the models [7, p. 13]. If the implementations are generated automatically from models to software code, then it is also a question of model driven development (MDD) [20, p. 45].

## 2.4 Software Architecture Description Languages

An architecture description language (ADL) is defined in standard ISO 42010 *"Systems and software engineering — Architecture description" as "any form of expression for use in architecture descriptions"* [31, cp. 4.5]. The standard sets some requirements for such a free-form expression, though. An ADL is expected to specify how to identify both

concerns to be expressed and the stakeholders of those concerns. In addition, an ADL is expected to specify architecture viewpoints and so called *"model kinds"* for framing the identified concerns. The specification requirement includes also correspondence rules related to the model kinds [31, cp. 6.3]. A model kind is model for certain type of modelling. For example, class diagrams and state transition models are model kinds [31, cp. 3.9]. Correspondence rules describe how model kinds should relate to each other, as in the following example: *"every task in task architecture shall execute a state machine"*. Binding the actual tasks and state machines is the correspondence, for instance: *"sm1 executes on task1"*. Correspondences indicate dependences, support traceability and express consistency in architectures [31, cp. A.6].

ADL's are used in both hardware and software architecture design. Hardware ADL's or *"machine description languages"* are used, for example, for describing processor architectures. The field of processor development especially gains benefit from the automatic design generation that the processor description languages utilize. Hardware ADL's focus on describing components, which differentiates them from modelling languages that are more used for describing behavior of an entire system. An example of such modelling language is the unified modelling language (UML) [38].

UML is commonly known modelling language and it has a large set of tools for system and software architects [39, cp. 1]. However, ISO 42010 refers to it only indirectly by declaring an ULM extension SysML as an ADL [31, cp. 4.5]. SysML uses many UML diagrams such as sequence, use case, state machine and package diagrams [31, cp. A.7]. The SysML extensions to UML are specified in SysML profile [40, Fig. 4.1]. An example of an extension in the profile is a block definition diagram, used for presenting a system as a decomposition of hardware, software and human elements. The block diagrams are based on UML class diagrams [7, cp. 2.12.1]. Another SysML extension is a requirements diagram, a presentation of system requirements in a graphical format [40, cp. 16.1]. SysML and UML can be applied in conjugation. An embedded system of hardware and software can be modelled with SysML and the finer details of software can be modelled using UML. Another UML profile named MARTE, can be used for modelling real-time behaviors of embedded systems [7, cp. 2.1]. MARTE extends the UML with timing diagrams and stereotypes for hardware resources and elements related to software concurrency, such as tasks [7, cp. 13].

The formal notations provided by ADL's promote automatic code generation and automated analyses of architectures. Yet, the usage of ADL's is not favored much [20,

p. 330]. According to a survey made in 2015, 41% of software architectures are done using UML. Only 12% of the respondents used an ADL alone. Few percent use some unknown methods and rest of the architecture descriptions mix UML and ADL's [41]. UML was originally developed by Rational Software, now part of IBM, and they have given a statement about the suitability of UML for describing software architectures. The results of the above-mentioned survey support this statement:

> *"Software architectures are specified by models. UML 2.0 provides a convenient set of modelling concepts suitable for directly capturing most interesting architectural patterns. UML 2.0 can therefore support a variety of architecture description languages (ADLs)."* [42, p. 63]

## 2.5 Software Architecture Verification

The final requirement for a software architecture of a medical device in IEC 62304 is that the software architecture has to be verified. A traceability analysis is recommend to be used for ensuring that software requirements are reflected in a software architecture. The emphasis is on verifying that the architecture implements risk control related requirements. In addition, verification should provide evidence that the architecture supports *"interfaces between software items and between software items and hardware"* and *"proper operation of any SOUP items"* [5, cp. 5.3.6]. Furthermore, software architecture should describe behavior of any software item that can affect to the behavior of any other software item or hardware. Modelling the behavior of a system is further implemented and the implementation depends on the architecture. Thus, the architecture have to be tested in order to demonstrate that the implementation of the system is safe. Verification of a software architecture is typically performed by conduction a technical evaluation [5, cp. B.5.3]. By evaluation, IEC 62304 means "*a systematic determination of the extent to which an entity meets its specified criteria*" [5, cp. 3.7].

Generally, software testing compares implemented functionality against requirements. However, software testing relates to architecture verification for the reason that software implementations are derived from the architectures of them. Software testing is typically divided to unit testing, integration testing, system testing and acceptance testing. Unit testing is used for testing individual software items, modules or *"units"* defined in the architecture. Integration and system testing are performed to a group of items co-existing as specified by the architecture and communicating with interfaces specified by the

architecture. Acceptance testing, or alpha- and beta-testing, is performed by internal and external customers. It may still reveal performance, security and other problems from an architecture, even though it is done in a very late phase of the development of a software or a system [20, cp. 19.2]. IEC 62304 mentions that integration and system testing do not have to be performed on *"full medical device"*. Instead, testing can be done also in a simulated environment [5, cp. B.5].

In addition to verifying that a system is implemented according to the architecture of it, the architecture itself should be evaluated. The architect should evaluate effect to desired quality attributes, for instance to performance and availability, every time when making a significant design decision. Peer-reviews can be organized for scenario-based analyses during design phase. Evaluations performed by outsiders are typically used after the design is finished. Full-scale evaluation can be conducted using a formal method, such as architecture tradeoff analysis method (ATAM). The ATAM method gathers stakeholders and relevant project leaders and members, including the architect, to scenario-based analyzing sessions, in which the goal is to identify the risks that architectural decisions may cause to the quality of a product. The formal ATAM process requires significant involvement from the participants. [20, cp. 21]

Architecture or design can also be verified by re-engineering an implementation to a reconstructed design and comparing that to the original design. Necessity for reconstruction may arise if traceability is weak between design and implementation. In regulated environments, reconstruction is used for updating documentation of legacy systems to the level that is compliant with related regulations. Another semi-formal verification method is discrete event simulation, in which a list of events are injected to a simulation of a system in order to estimate, for example, system's failure rate. [35, cp. 14]. State machine diagrams are often used for modelling behaviors of embedded systems and state machines indeed are event driven: events trigger transitions between states [7, pp. 100-101]. ISO 14971 references discrete event simulation and ISO 26262, a functional safety standard for road vehicles, recommends simulation to be used for verification of software architecture [35, pp. 190-191]. Formal design verification methods describe systems mathematically and support automatic code generation [35, cp. 15]. However, IEC 62304 especially does not require any formal methods to be used in software requirements verification [5, cp. 5.2.6].

# 3 Materials and Methods

The background information presented in the previous chapter has significant relevance for the company as it operates in the environment explained and has an ongoing development project to which this study also contributes. The company's interest towards the topic can be summarized to a research question as follows:

> *How do the changes in the regulated environment affect to the development processes of IVD medical devices and how to design a software system architecture of such in the manner that makes it compliant with the related regulations?*

Materials and methods of this study consist of company material and analysis method of those, methods for developing and describing a software system architecture in the project mentioned and of evaluation method of the results of the work.

## 3.1 Company Material

The company material consist of standard operation procedures (SOPs) related to product development of both general-purpose laboratory equipment and IVD medical devices. Some of the SOPs were written before the medical device manufacturing industry went through the regulation changes and it was assumed that they might not have been updated since. Thus, it was anticipated that some changes to the SOPs were required. The SOPs mainly refer to ISO 13485 and IEC 62304. The analysis of the SOPs was performed against those and other related standards and regulations. The company material is not presented in detail. However, the recommendations on how to update the SOPs are included in the results of this study. The focus in the analysis was set on software development, although other subject matters were also addressed.

In addition to the analysis of SOPs, the company requested a study of the differences between requirements for manufacturers of components used in medical devices and requirements for manufacturers of medical devices. In its essence, the company question is how the new regulations are interpreted in the case where the company is a component supplier for an IVD system. The method of investigating the issue was a literature review of the medical device regulation (MDR) and the IVD medical device regulation (IVDR).

## 3.2 Methods for Architecture Design and Description

The selected strategy for developing the software system architecture in the company project was to meet the requirements of IEC 62304 and especially of the chapter "*Software architectural design*" [5, cp. 5.3]. Several quality goals were set for the outcome in addition to the requirements of the IEC 62304. The main quality attributes pursued were availability, interoperability, maintainability and testability. In other words, the working system should be reliable and the internal and external communication means should be usable and well documented. Furthermore, the architecture should be understandable in that degree that any changes to the system can be made easily and the running system should provide means to verify the architecture.

The architecture design principles were selected to follow and highlight the best practices found from the embedded system software development design style of the company. The main design principles were set to modularity, event-driven programming and model based development. Several architectural patterns were selected for the design to support the principles. Along with patterns, various tactics were selected to be used to achieve the quality requirements set in the architectural strategy. The design decisions are elaborated in the results.

The architecture description model was selected to follow the Kruchten's 4+1 view model [32]. The naming convention is slightly different from the model and the model was extended with a system overview named *"Executive View"*. Following is the list of the views.

- *Executive View*
- *Functional Description*
- *Development View*
- *Task Architecture*
- *Deployment View*
- *Scenarios*

UML diagrams were selected as a principal notation for the architecture description. The executive view is for the viewpoint of business and marketing and it was allowed to use a visually more appealing style of description than plain UML. Documenting the views was based on view templates presented in the book *"Software Architecture in Practice"*.

[20, p. 347]. Additional inspiration to the description were taken from SysML examples of Object Management Group [40, Annex D].

## 3.3 Evaluation Method

The study is closed with a qualitative evaluation of the outcome. The evaluation was performed by conducting a series of stakeholder interviews. The interviewees were a selected group of persons from different parties such as business, project management, project team, quality department and from validation and verification. Before the interviews, the interviewees were given a work-in-progress version of the thesis to read.

The interviews were conducted according to the template presented in the Appendix 1. The interview plan was to record the role and concerns of the interviewees and then let the persons speak freely about the thesis before asking them questions that were more detailed. The interviewee profiling included examination of attributes such as title and relation to the project or/and to the thesis. In addition, the viewpoint and concerns of the interviewees were recorded. Asking the more detailed questions after the free speech was intended to ensure that the feedback of all interviewees discussed the subject homogeneously. The interviews were recorded to audio/video and the recordings were deleted after analyzing them. The interviewees are presented anonymously in the results. The interviewees were rewarded with a small gift after the interviews.

# 4 Results

The first part of the results of this study is the analysis of standard operating procedures for product development, especially for software development, of a product development team of the company. The analysis is followed with an investigation of requirements for component suppliers of IVD systems. Second part of the results consist of IEC 62304 compliant architecture description. The compliancy is subjective, official assessment is not within the scope of this study. In addition, the results include an analysis of interviews that were conducted for the purpose of qualitative evaluation of this study.

## 4.1 Analysis of the Company SOPs and Recommendations for Updates

The highest level of the company SOPs under analysis focuses on product development processes and responsibilities for the activities defined in the processes. Both ISO 13485 and the basis of it, ISO 9001, are referenced. The reason for the company mentioning the both seems to be the fact that the SOPs are targeted for developing of both IVD medical devices and general purpose or research-use laboratory equipment and instruments. The ISO 9001 standard, or *"Quality management systems -- Requirements",* focuses on customer satisfaction and continuous improvement [43, pp. 2-3]. Whereas, the ISO 13485 emphasizes safety and performance instead [3, cp. 0.1]. The high-level SOPs of the company move the responsibility of the quality management system selection onwards to individual product development projects. Some of the related SOPs contain references to standard versions ISO 9001:2008 and ISO 13485:2012. A recommendation for the company is to update the references to address the latest versions ISO 13485:2016 and ISO 9001:2015.

Not many of the changes in the ISO 13485:2016 are closely related to development phase of a medical device. However, one new requirement in the 2016 version explicitly demands that manufactures have to maintain a *"design and development file"* for including or referencing design and development documents [3, cp. 7.3.10]. The company SOPs refer to FDA's design history file (DHF) for the same purpose. A recommendation is to update the SOPs to address the design and development file also. The 2015 version of the ISO 9001 lays focus on *"risk-based thinking"* and is structured to support easier use of multiple quality management systems [43, p. 8]. A detailed analysis of the ISO 9001:2015 is not within the scope of this thesis. A recommendation

for the company is to consult accountable persons from quality department for investigating if it has any effect to the SOPs.

The company SOPs related to software development are well aligned with IEC 62304:2006. All the required software life cycle processes are defined and documented. A traceability matrix between the SOPs and IEC 62304, FDA regulations and other related standards is provided. Software architecture design is required and peer reviews are defined as the main verification method of architectures. Software architecture description is required regardless of the software safety classification. It is notable, that there is no clear differentiation in the defined activities between development of general purpose and IVD products. The company should note that the requirements in the IEC 62304 standard differ between different safety classes and that the safety classification is relevant only to a medical device software.

A recommendation for the company is to increase knowledge on the issue of what is exactly required from product development projects when developing IVD medical devices instead of general-purpose instruments. Particularly, clarifying the issue of what kind of products actually are IVD medical devices, what are not, and why some type of products cannot be, could aid product development in the company. These topics are thoroughly discussed in EU's guidance documents MEDDEV 2.14/1 *IVD Medical Device Borderline and Classification issues* and MEDDEV. 2.14/2 *IVD GUIDANCE: Research Use Only products*. The guidance documents interpret the repealed IVD directive, but similarities can be found from the new regulation also.

According to the MEDDEV 2.14/1, the main feature of any IVD product is that it produces information for medical purposes from samples derived from a human body. In addition, an IVD product must be intended for use by the manufacturer of the product for IVD medical purposes. However, the manufacturer has to have adequate amount of *"technical, clinical and scientific data"* that justifies the announced IVD medical intended use. Accessories for IVD devices are a special type of products. An accessory does not have a medical purpose, but it is intended by the manufacturer of it to be used with a particular IVD device. If the accessory enables the intended purpose of the particular IVD device, then it is *"treated as in vitro diagnostic medical device"*. From another perspective, general-purpose laboratory products are not IVD devices unless they are intended and qualified to be [44, cp. 1]. This is stated very clearly for sample preparation devices:

> *"Products used in vitro in the preparation of samples that have been obtained for examination are considered neither as IVD nor as accessories and fall outside the scope of the Directive unless, based on their characteristics, they are specifically intended for a particular IVD test. The validation of this specific combination shall be clearly documented in the technical documentation."* [44, cp. 1.4]

MEDDEV 2.14/1 list various examples of borderline products and refers to document *"Manual on Borderline and Classification in the Community Regulatory framework for medical devices"* for more examples. The manual is last updated in December 2017. One of the latest product types specifically excluded from the IVD devices is *"microplate washer"* [45, cp. 7.5]. The example list of general purpose non-IVD products in MEDDEV 2.14/1 contains equipment such as *"general purpose pipettes"*, *"empty ELISA plates"*, *"DNA and RNA extraction kits that only provide a specimen without an intended IVD detection combination*", *"incubators"*, "*spectrophotometers*" and *"ELISA readers providing raw data which is not readily readable and understandable by the user (e.g. peaks, OD)"* [44, cp. 1.4].

It is possible that a product is intended to be used for both IVD medical purposes and other purposes. Then the product must comply with IVD requirements. Nonetheless, the IVD use must qualify the definitions of an IVD medical device. Furthermore, if a product has an IVD medical intended use, then it cannot be placed on the market as *"research use only"* (RUO) product [44, cp. 1.1]. RUO products are used for instance in pharmaceutical or other basic research. They may handle specimens derived from human body, but they do not use the specimens in the manner that is characteristic for an IVD medical device [46, cp 0.5]. For avoiding the *"misuse"* of RUO products in clinical laboratories, the European Commission has stated that a RUO product must not have a *"medical purpose or objective"* [46, cp 0.2]. In addition, RUO products are not to be used as *"devices for performance evaluation"*. Devices for performance evaluation are products becoming to IVD medical devices after performance evaluation studies [46, cp 0.4].

The IVD regulation concedes to above discussed MEDDEVs. The IVDR does not apply to general-purpose laboratory products or research-use only products unless they are specially intended to be used for IVD diagnostics [2, Article 1]. Ultimately, it is the medical device coordination group (MDCG) that decides if a product is within the definition of an IVD medical device [2, Article 3]. As it comes to accessories, the IVDR requires that they

have to be classified separately from the IVD devices utilizing them [2, Annex VIII]. FDA has recently changed its approach to accessories towards the same thinking as in EU; the risk classification of an accessory for a medical device in U.S. market is to be classified separately from the parent device [47, p. 3]. The question of accessories relates to a particular goal of this study, which is to clarify the usage of non-medical components in a medical device. The topic is discussed in the chapter 4.2.

On the subject of knowledge increase, another recommendation for the company is to provide more guidance for IVDMD product development on formulating an intended use, classifying a risk class and classifying a software safety class. This is important for software development for the following reason: if a development project defines an IVD medical use for a product containing software, then an IEC 62304 compliant software risk management process has to produce a safety classification for the software. It should be noted that software of an IVD device, or of an accessory for IVD device, has always the same risk classification as the device in which the software is running [2, Annex VIII]. However, the software safety classification is not the same thing as the device risk classification. The software safety classification is explained in chapter *2.2 Software System Architecture Development in the Regulated Environment.*

A risk analysis of a medical device performed according to ISO 14971 should identify hazardous situations, which may or may not *"arise from a failure of the software"*. A recommendation for the company is to make a note that a lot of IEC 62304 requirements can be omitted if an unacceptable risk caused by the hazardous situations can be mitigated with measures external to the software. These requirements include, among others, software architecture design, software unit testing, software integration testing and most of the software risk management process activities [5]. ISO 14971 contains a guidance especially for risk management of IVD products. A recommendation for the company is to use the guidance as an inspiration for the SOPs to illuminate, for example, how software can be involved in a hazardous situation where medical treatment is delayed or incorrect because of IVD device malfunction [48, Annex H].

Optimizing the implementations of activities of IEC 62304 life cycle processes based on software safety classification makes no sense if the product under development has no medical intended purpose. A recommendation for the company is to clarify in the SOPs that products outside of medical context should be developed within quality management system compliant with the standard ISO 9001:2015 *"Quality management systems – Requirements"*. Software development for such product should address the standard

ISO 90003:2014 *"Software engineering -- Guidelines for the application of ISO 9001:2008 to computer software"*. Last recommendation for the company is to note that also IEC 62304 recommends complying with ISO 90003 [5, cp. B.2]. Finally, all the recommendations given in this chapter are summarized in the Table 1 below.

Table 1. Summary of recommendations for updates to standard operation procedures.

| Recommendations for Updates to Standard Operation Procedures | |
|---|---|
| **1. Product Development SOPs** | |
| 1.1 | Update references to standards to address the latest versions. |
| 1.2 | Consult accountable persons for gap analysis between ISO 9001:2008 and 9001:2015. |
| 1.3 | Advice non-medical product development to comply with ISO 9001:2015. |
| 1.4 | Make a reference to design and development file as in ISO 13485:2016. |
| 1.5 | Clarify differences between requirements of IVD and non-IVD development. |
| 1.6 | Create guidance for formulating intended use and classifying a risk class for medical products. |
| **2. Software Development SOPs** | |
| 2.1 | Advice non-medical device software development to comply with ISO 90003:2014. |
| 2.2 | Advice medical device software development to consider complying ISO 90003 as it is recommended by IEC 62304. |
| 2.3 | Create guidance for assigning of software safety class according to IEC 62304 for medical device software. |
| 2.4 | Advice avoiding many IEC 62304 requirements by always pursuing software safety class A. |
| 2.5 | Advice IVDMD software risk management to lookup guidance from IVD specific examples in ISO 14971. |

## 4.2 Components Supplied to Medical devices

This chapter presents results of an investigation on the topic how the requirements for manufactures of components used in medical devices differ from the ones set for the manufacturers of the medical devices. Including the topic to the goals of this thesis was especially requested by the company. One of the roles of the company is a component supplier for IVD systems and for that reason, an interest of the company is to understand how the IVDR addresses the issue.

As explained in the chapter 4.1, an accessory for an in vitro diagnostic medical device is regulated by the IVDR only if the manufacturer of the accessory specifically intends the accessory to be used together with an IVD medical device. It was also explained that even then some products might not to be considered as accessories. There are other options for including components in IVDMDs, though. As is the case with MDs also.

By definition, an IVD system is interpreted as IVD medical device in the IVDR [2, Article 2]. On the contrary, the MDR does not include term *"system"* in the definition of a medical device [1, Article 2]. Instead, the MDR recognizes *"systems and procedure packs"* as combinations of MDs or MDs and IVDMDs. A system in the MDR context may also include *"other products"* if *"their presence"* in the system is *"justified"*. It is then the responsibility of the system manufacturer to perform *"appropriate internal monitoring, verification and validation"* for the *"other products"* also. However, a combination loses its system status if the parts of it are *"not compatible in view of their original intended purpose"*. Then the combination is treated as medical device and is required to conform *"the relevant conformity assessment procedure"* of such [1, Article 22].

A closest equivalent for MDR system in IVDR is a *"kit"*. It is defined as a *"set of components that are packaged together and intended to be used to perform a specific in vitro diagnostic examination"* [2, Article 2]. Even so, the IVDR articles where the *"kit"* is mentioned do not discuss non-medical products. For instance, it is required that each individual device, meaning IVDMD, in a kit have to comply with requirements of the IVDR [2, Annex I cp. 20.4 (s)].

Nevertheless, non-medical components are mentioned in the IVDR. For example, instructions for use has to contain information on the general-purpose equipment that is used in combination with an IVDMD. The required information includes key performance characteristics of the equipment and *"any known restrictions to combinations of devices and equipment"* [2, Annex I cp. 20.4 (j)]. Furthermore, a *"summary of safety and performance"* documentation of an IVDMD have to contain descriptions of products used in combination with the device [2, Article 29]. Moreover, whenever connecting an equipment to an IVDMD for intended combined operation, then the manufacturer of the combination has to prove the compliance of the combination against the general safety and performance requirements of the IVDR [2, Annex II cp. 6.5].

The general safety and performance requirements in both MDR and IVDR require that if the intended use of a device require the device to be used in combination with other

equipment, then the combination of devices must be safe to use and perform as specified. [1, Annex I cp. II 14], [2, Annex I cp. II 13]. In addition, both regulations require that technical documentation of a device have to include descriptions of non-medical products that are intended to be used in combination with the device [1, Annex II cp. 1], [2, Annex II cp. 1]. Moreover, the technical documentation including the non-medical product descriptions may have to be assessed as part of a conformity assessment procedure [1, Article 52], [2, Article 48].

Manufacturers of MD and IVDMD devices are also obligated to address *"selection and control of suppliers"* in their quality management systems [1, Article 10], [2, Article 10]. Following are examples of the IVDR requirements related to suppliers: *"announced and unannounced inspections of the premises of suppliers"*, *"identification of all sites, including suppliers, where manufacturing activities are performed"* and *"audit the control of processes on the premises of the manufacturer's suppliers"* [2].

In summary, a manufacturer that supplies a component to an IVD system is under IVDR only if the component is IVDMD or an accessory of an IVDMD. If the component is general equipment, then the component manufacturer should have to be prepared to provide descriptions of the usage and performance for the IVDMD manufacturer. In addition, the component manufacturer can expect that the IVDMD manufacturer or the notified body of the IVDMD manufacturer might want to perform inspections and audits of the premises and processes of the component manufacturer.

## 4.3 IEC 62304 Compliant Software Architecture

This chapter introduces an architecture of a software system related to a product development project of the company. The description of the architecture is presented in a generic level for the sake of comprehensibility. Partly the reason for abstracting the details is confidentiality of the company data. Nevertheless, the architecture description presented in this chapter can be used as an example when designing medical device software system architecture. That is because, in addition to the description, the requirements in the IEC 62304 for a medical device software architecture are addressed chapter by chapter.

### 4.3.1 Architecture

The first requirement in the IEC 62304 for an architecture is that the architecture has to be constructed from the software requirements. The architecture description has to present a structure of the software and list all items of the software [5, cp. 5.3.1]. A software item is for example a collection or a part of source code. [5, cp. 3]. The overall architecture requirement is mandatory only for software systems with software safety classification B or C [5, cp. 5.3.1].

The software requirements of the company project were derived from various sources and documented in a requirements management system. Voice of customers is presented in a design input document, created by a marketing team. Technical and functional requirements were constructed by a team of system designers, architects and product owners. Quality and regulatory affairs were also involved in specifying the requirements. The software requirements include requirements for the software system architecture. The design principles selected for the architectural design were modularity, event-driven programming and model-based design. The principles were added to the requirements and clarified to a more detailed level.

Modularity requirement demands that program code of software items is separated into modules and the modules should hide as much as possible of their internals from the public interfaces to the modules. Event-driven programming means that the execution of an application code is done in event or message handlers, which never block the execution. Blocking means that the handlers would pause the execution of themselves and would wait a synchronization from hardware events or from other tasks of the underlying OS in order to continue. The execution time of the non-blocking handlers must be kept as short as possible.

Model based design principle means that the behavior of the software systems is designed using UML diagrams. Mainly, modelling means utilizing of hierarchical state machine diagrams. State machines are implemented in an event-driven fashion using a specific proven framework. Implementation of the state machine diagrams has to be verifiable from both the real system and using unit testing and simulation. In addition, modelling means utilizing of activity diagrams where applicable. Preferably, activity diagrams are used to describe logic of complex algorithms.

The requirements are reflected by a software architecture, which is documented as a software architecture description. A standard operating procedure for software development process of the company requires that the software architecture description must be included into a software design description (SDD). Standard IEC 12207, from which the IEC 62304 is derived, recommends that the SDD should comply with standard IEEE 1016 *"Standard for Information Technology—Systems Design—Software Design Descriptions"* [5, cp. C.6], [49, Table G.1]. The actual requirement in the IEC 12207 is that an output of a software detailed design process is a detailed description of the design [49, cp. 7.1.4.2]. This thesis refers to, but does not include, the software design description of the related company project. The model of the software architecture description of the company project is presented in the Figure 14.

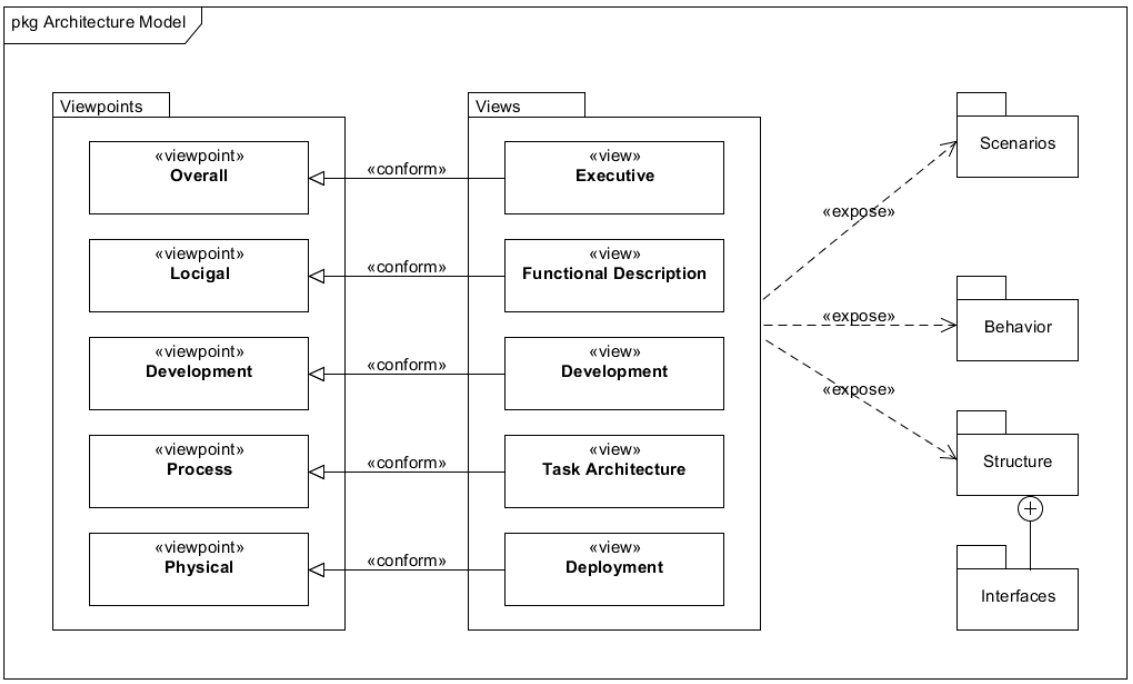


Figure 14. Software architecture description model. Notation: UML package diagram with SysML stereotypes.

The description model of the software system architecture of the product under development in the company project is presented in the Figure 14. The model is a UML package diagram, as is indicated by the abbreviated frame name *"pkg"* in the diagram frame heading. The *"views"* package of the software architecture description model contains five different views, each of them conforming to concerns of a viewpoint. Furthermore, the views expose behavioural and structural models of the software architecture from the model packages. The diagram utilizes SysML stereotypes *"conform"* and "*expose"* for the relations of the packages. The diagram violates SysML

standard purposely, in order to abstract complexity. Similarly, as is the case with *"conform"* relations, the starting edges of the *"expose"* relations should be individual view elements [40, cp. 7.3.2.3].

The view model is based on Kruchten's 4+1 model. Executive view is for executive summary and similar purposes. It provides a general understanding of the system. The functional description is the logical view. It describes the interfaces and the behavior of the system. The development view describes the module decomposition, layers and the development environment. The task architecture is the process view. The deployment view presents the physical environment to which the software items are deployed. The fifth or the "+1" view of the Kruchten's 4+1 model, the scenarios, is also a model package. Scenarios package contains use cases of the overall system. Interfaces are part of the structure package, which is indicated by the "circle-plus" package membership notation in the diagram [39, cp. 7.4.4.1].

The viewpoints and views are documented partly using and partly extending the view template presented in the book *"Software Architecture in Practice"* [20, p. 345]. Following are the chapters of the modified template.

- *Viewpoint*
- *Primary Presentation*
- *Element Catalog*
- *Context Diagram*
- *Variability Guide*
- *Rationale*

The extension to the template is that for each view, a viewpoint is documented. Adding the viewpoints makes the template more compliant against view description requirements of standard ISO 42010. This gap between the original template and the ISO 42010 is recognized by the authors of the aforementioned book also [20, p. 360].

The documentation of the views of the software system architecture of the company project utilizing the template is presented in Appendix 2. It is to be noted that not all sections of the template are relevant to every view. Thus, some sections are marked as *"not applicable"* (NA) in the view documentation. In this chapter, the sections of the

template are discussed using the elements of the architecture description of the company project as examples.

*"Viewpoint"* of a view presents the stakeholders of the view and their concerns towards an architecture. In the company project, the stakeholders and concerns of the viewpoints are documented using SysML notation. Stereotypes *"stakeholder"* and *"viewpoint"* are used in the headers of the block elements. Attributes *"concern"*, *"purpose"* and *"stakeholder"* are used inside blocks. The *"stakeholder"* attribute in the *"viewpoint"* block binds the *"viewpoint"* block to the *"stakeholder"* block. There are various other attributes in the SysML for views and viewpoints that are not used in the case architecture, as they are relevant only in an actual SysML modelling environment. The reason why the partial convention was selected was to demonstrate and practice a description style that is capable of capturing relatively abstract definitions to a more standardized format. As an example, Figure 15 presents a viewpoint selected from the Appendix 2.

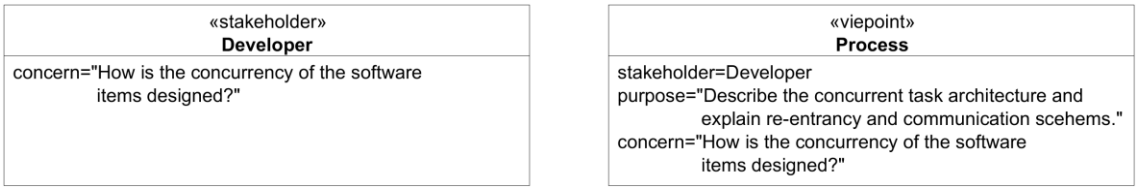


Figure 15. Viewpoint *"Process"* to the view *"Task Architecture"*. Notation: SysML.

The viewpoint in the Figure 15 is the *"Process"* viewpoint. It is conformed by the view *"Task Architecture"*, as demonstrated in Figure 14. The stakeholder of the viewpoint is a *"Developer",* who is concerned how the designed process or task architecture should be implemented. The purpose of the viewpoint is to address the concerns of the developer. Other stakeholders of the different views of the company project are *"product manager"*, *"user"* and *"integrator"*. The *"integrator"* is a stakeholder of the viewpoint *"Physical"*. The concern of the *"integrator"* is *"how are the software items deployed to the device?"* That concern is addressed in the view *"Deployment".* Another example of a concern is the one that the *"product manager"* has: *"Is the system going to realize itself as intended?"* That concern is addressed by *"Overall"* viewpoint, conformed by the view *"Executive View"*.

*"Primary Presentation"* is the chapter of the template that presents a main diagram of a view. The notation key of a main diagram is obligatory in the primary presentations, as is with all other diagrams in the view documentation. All the primary presentations in the view documentation of the company project use UML diagram notation, expect the

primary presentation of the view *"Executive View"*, which uses informal notation. The reason for the informal notation in the view is to address the stakeholders of the view with a presentation, which does not require them to have knowledge of UML details. The main diagram of the "Executive View" is presented in the Figure 16.

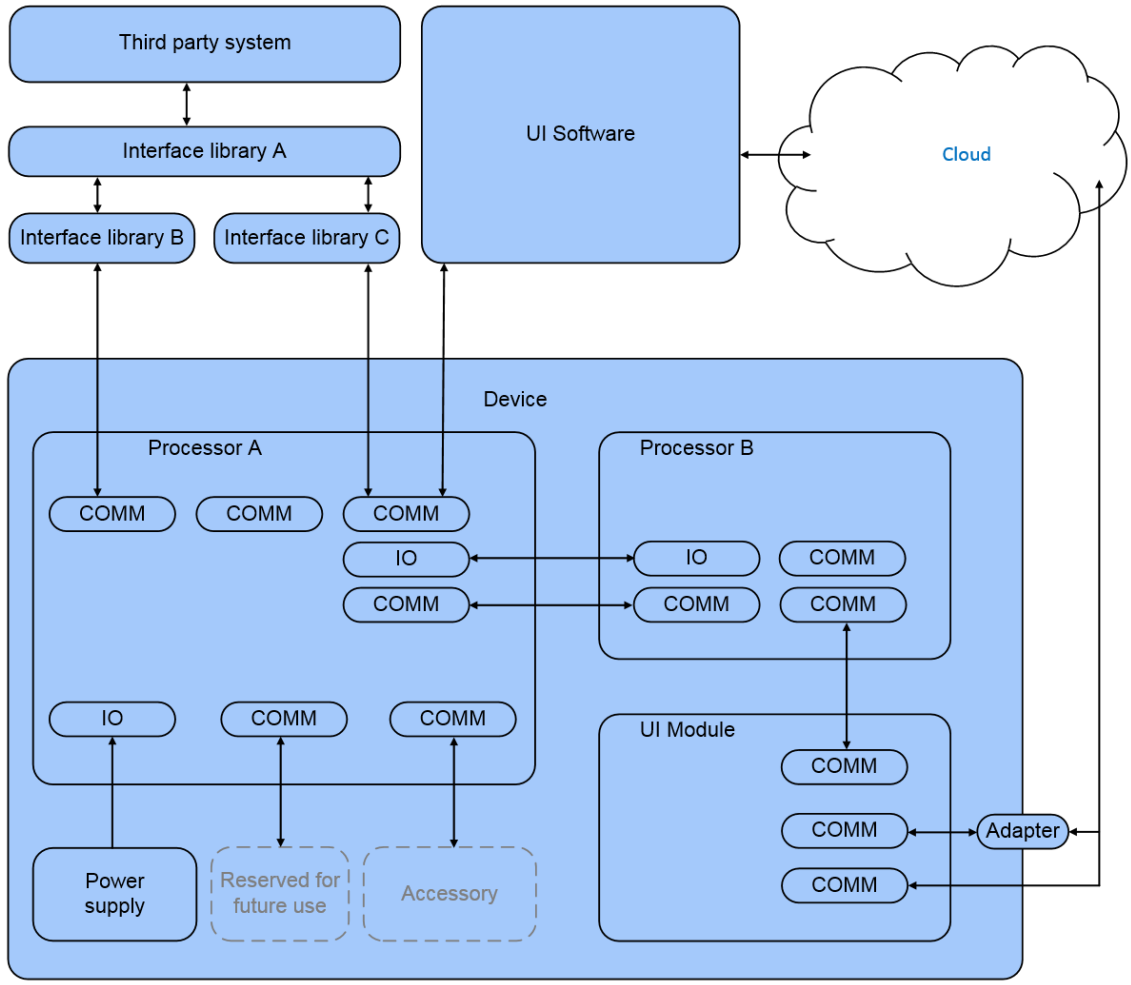


Figure 16. Executive summary level system diagram. Notation: Informal.

It should be noted that the main diagram of the *"Executive View"* in the Figure 16 combines hardware and software elements, and is thus representing system architecture. In fact, IEC 62304 compliant software system architecture have to describe hardware. First, software requirements of a medical device have to address topics such as *"platform, operating system, hardware, memory size and processing unit"* in "*functional and capability requirements*" [5, cp. 5.2.2]. Secondly, the interface architecture part of the software architecture has to describe hardware interfaces [5, cp. 5.3.2]. In addition, hardware requirements of SOUP items must be specified in the software architecture [5, cp. 5.3.4]. Furthermore, when verifying the architecture, the

support from the architecture for interfaces between software and hardware have to be verified [5, cp. 5.3.6]. Finally and for software items with software safety classification C only, detailed designs of interfaces of software units have to include interfaces to hardware [5, cp. 5.3.6]. The details of the primary presentation of the *"Executive View"* are visible in the Appendix 2. Another type of a main diagram is presented in the Figure 17. The diagram is from the primary presentation of the view *"Task Architecture"*. It is lacking any hardware elements and is using a UML notation.

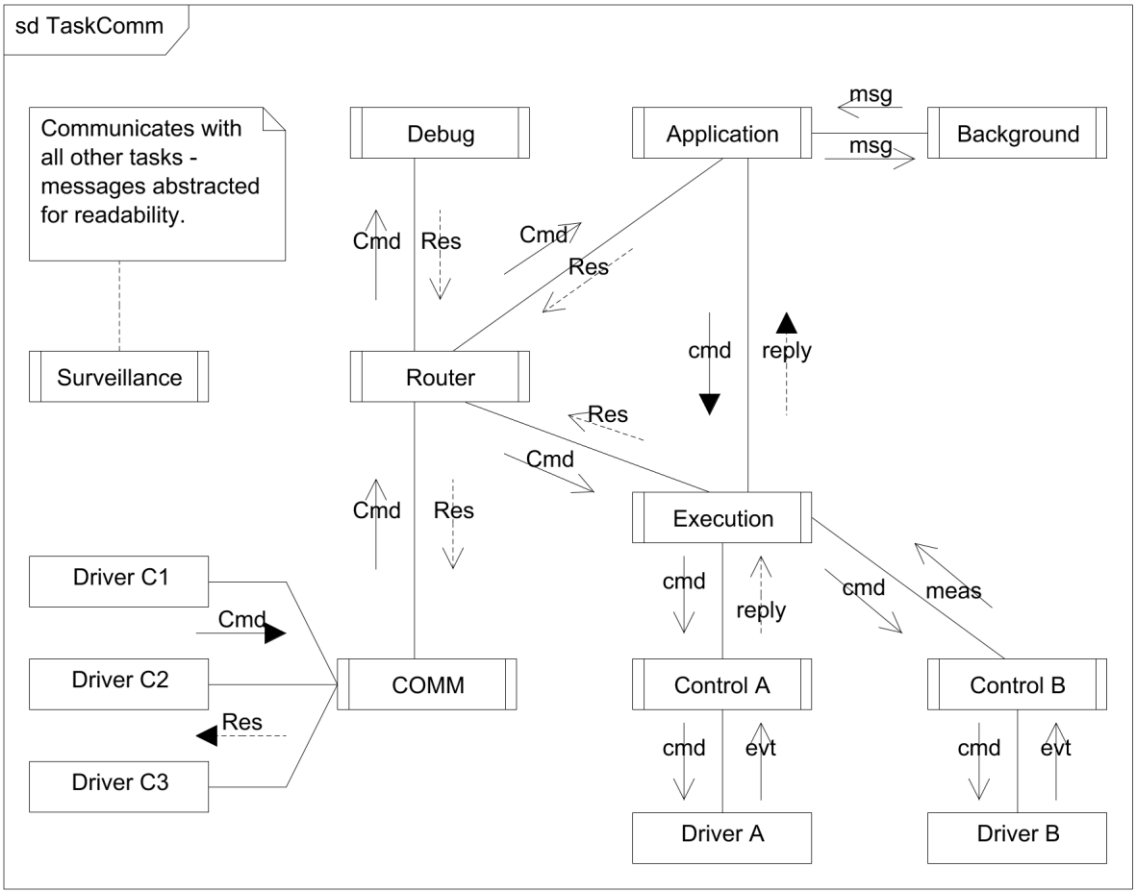


Figure 17. Communication scheme between tasks of *"Processor A"* software item. Notation: UML communication diagram.

Notation for the presentation in the Figure 17 is UML communication diagram, which is a variation of UML interaction diagram. The diagram illustrates the internal control and messaging architectures of the software item in question. In the diagram, the software item is controlled from external systems using synchronous command-response messaging. This is indicated in by closed black arrowheads. Interprocess communication inside the software item uses asynchronous (open arrowheads) messages in addition to synchronous messages. UML active object symbols are used for tasks. Drivers (passive objects) use plain box notation. Drivers send spontaneous event messages to the tasks.

The events are generated in hardware interrupt handlers. Systems attached to the external interfaces are notified by events as well, put those are excluded from the diagram for the sake of readability. The rest of the main diagrams of the primary presentations of the views not discussed yet in this chapter use also UML diagrams. The view *"Development"* utilizes package diagram to demonstrate module decomposition of the software items. A deployment diagram is used in the view *"Deployment"* for mapping the software items to a physical environment. The view *"Functional Description"* presents interfaces of the system in a component diagram.

*"Element Catalog"* is the next chapter in the view template. The purpose of it is to describe details of elements presented in a *"Primary Presentation".* The *"Element Catalog"* is not merely a list of elements with short descriptions. Instead, it may present or reference detailed descriptions of interfaces, properties and behavior of elements in a view. An example of contents of an element description is presented in the Figure 18.

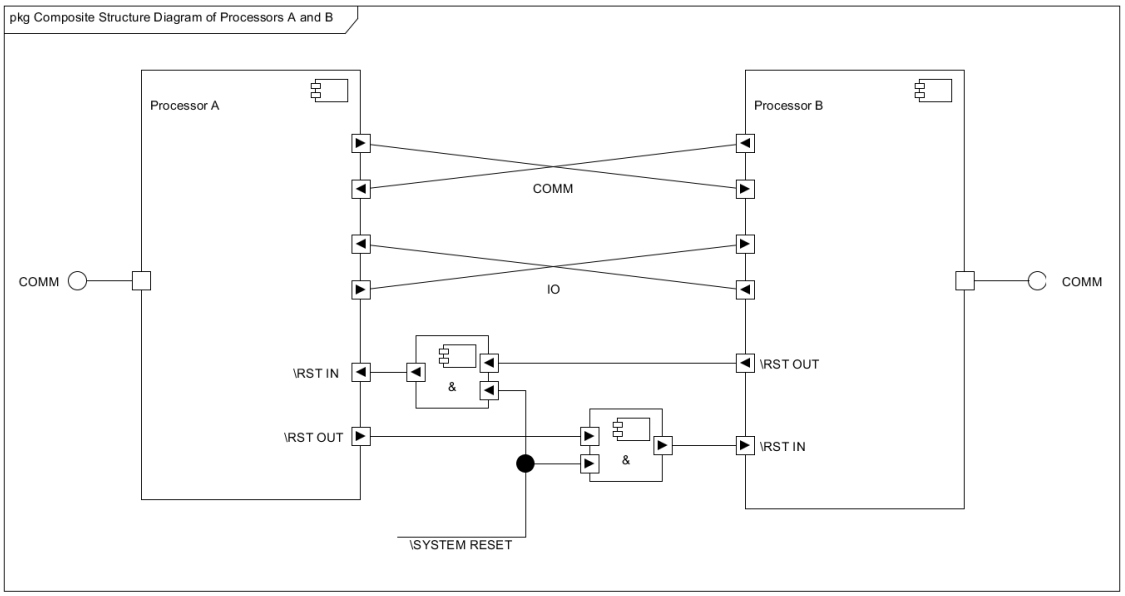


Figure 18. Interconnections of processors A and B. Notation: UML Composite Structure Diagram.

The example in the Figure 18 is taken from the element catalog of the view *"Executive View"*. It is an UML composite structure diagram, presenting how an essential architecture decision of the company project is designed in the system level – two computational units are capable of cross reprogramming each other's using the interconnection scheme presented in the diagram. Another example from the company project is a behavioral description from the element catalog of the view *"Functional description"*. The catalog contains an UML model of a hierarchical state machine, which is copied to the Figure 19.

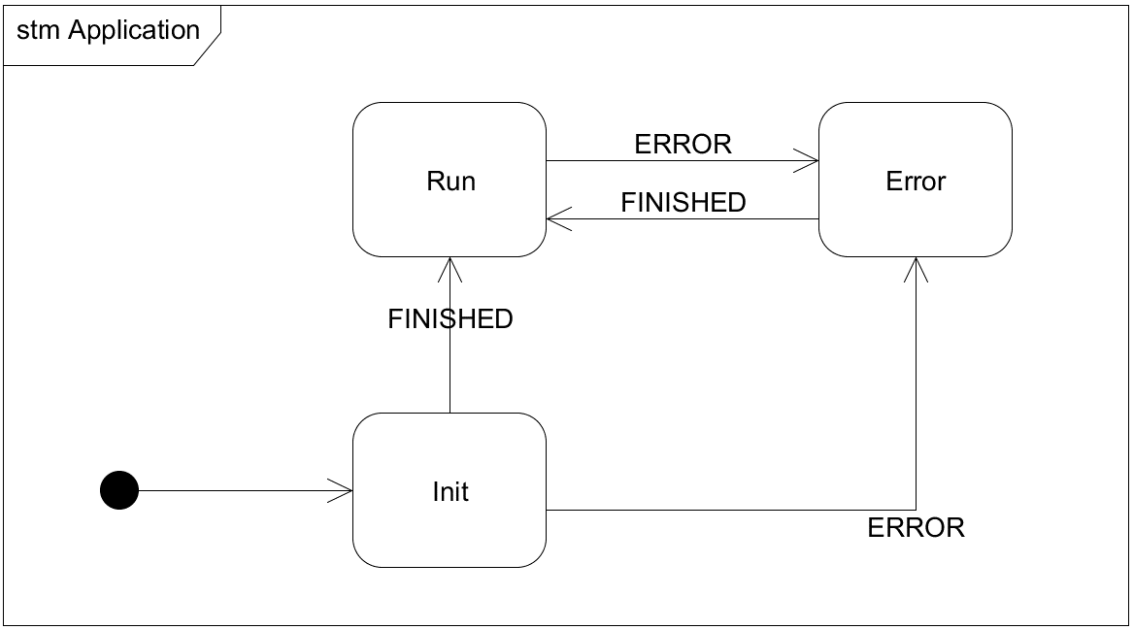


Figure 19. Application task state machine. Notation: UML state machine diagram.

The state machine diagram in the Figure 19 describes behavior of an element *"Application"*. The diagram is intuitive and very simple. Only three states are presented. However, the behavior package from witch the diagram is exposed by the view contains deep hierarchy of sub-states for each of the three main states. Every one of those sub-states are similarly modelled using the UML state machine diagram notation. In this case, the element catalog is an approachable entry point to a high-detail description, which is the behavior package. The behavior package of the software architecture of the company project contains also activity diagrams, which are another type of UML diagrams suitable for describing behavior. The commonly known synonym for activity diagram is flowchart. The activity diagrams are often used to model and describe complex algorithms. Descriptions of algorithms can be used to comply IEC 62304 requirement for software detailed design of un-dividable software units, meaning modules or functions [5, cp. B.5.4]. The detailed design of software units is required if the software safety classification is C [5, cp. 5.4.2].

The view template continues with chapter *"Context Diagram"*. The chapter is for presenting a part of a view or a view element in the middle of its environment or context. An example of a context diagram of the software architecture of the company project in the Appendix 2 is selected to the Figure 20 below.

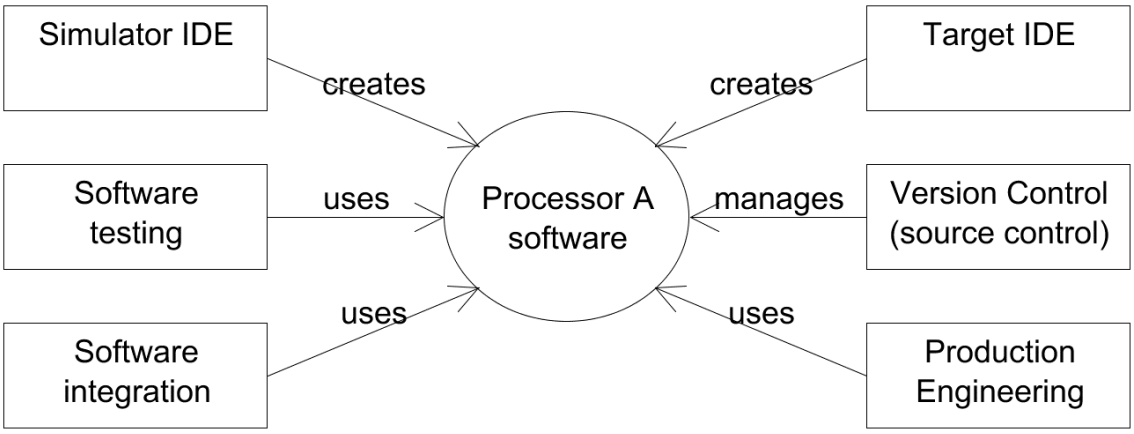


Figure 20. Context of *"Processor A"* software item development. Notation: Informal.

The example in the Figure 20 is taken from the description of the view *"Development"*. It describes the interactions that are related to the development and usage of a software item. The software item *"Processor A software"* in the middle of the diagram is surrounded by actors, which are either contributing to the creation and management of the item or using it as an input for other processes. An item can be presented in other contexts also. For instance, view *"Functional Description"* presents the software item *"Processor A software"* in the middle of its interfaces. The notations of the context diagrams are typically informal. However, some context diagrams of the company project use also SysML internal block diagram for the notation.

*Chapter "Variability Guide"* follows the chapter *"Context Diagram"* in the view template. The purpose of the chapter is to express how much there is leeway for the designers and developers when implementing the design. In addition, it addresses options, scalability and other topics, which are relevant after development phase also. The style of the chapter is a list of rules. Following are examples of variability rules of the software architecture views of the company project in the Appendix 2.

- *Executive View:* Accessories and future reservation have to support the physical properties of the interfaces.
- *Functional Description:* The behavioural models have to be implemented as modelled.
- *Development View:* Trivial tasks do not have to implement state machines.
- *Task Architecture:* New tasks should follow the same principles that are used for designing the initial control tasks.

- *Deployment View:* Only one temporary sequence can exists at a time in the device.

The last chapter of the view description template is the *"Rationale"*. It is a chapter of free text discussing justifications for design decisions of a view. For example, in the company project, rationale of *"Executive View"* justifies the selected combination of computational modules with increased availability and scalability. Another example can be found from the view *"Functional Description",* in which behavioral modelling is justified with better maintainability and testability. To continue with one more example, view *"Deployment"* justifies using tactic *"prevent reverse-engineering"* with increased security. In fact, listing tactics and patterns used in a view is the another purpose of the rationale.

Several architectural patterns are addressed in the view descriptions of the software architecture of the company project in the Appendix 2. Multi-tier pattern is used in the *"Executive View"* for allocating a user interface software item to a separate computing unit. Layering is used for abstracting hardware and interfaces and for dividing functionality based on required real-time performance. Layers are addressed in both the *"Development View"* and the "*Task Architecture"*. The kernel pattern is employed by using a micro-kernel, which is visible in the *"Development View".* The "*Task Architecture"* utilizes both hierarchical control and distributed collaborative control patterns in its control architecture. Furthermore, the view mixes asynchronous and synchronous messaging patterns in its internal and external communication schemes. The internal messaging scheme utilizes messaging services of the real-time operating system and is thus being essentially asynchronous. However, the control scheme includes synchronous command-reply messaging protocols between the tasks.

Along with patterns, various tactics were selected to be used in the views of the software architecture of the company project in order to achieve the quality requirements that were set in the architectural strategy. For availability, the tactics include *"send heartbeats"* and *"use watchdog"*. The heartbeat messages inform the receivers of them that the connection and the sender are viable. Heartbeat tactic is used in the view *"Functional Description"* in which the interfaces are described. In addition, the view elevates interoperability by utilising tactic *"produce well defined interface documentation"*. Watchdog is a feature of a surveillance task described in the *"Task Architecture".* A watchdog is a hardware mechanism used for rebooting systems if execution of an application code becomes irrational.

For maintainability, the main tactic was set to *"describe behaviour by modelling state machines"*. This is addressed by the view *"Functional Description"*. As is tactic *"provide debug interfaces",* which supporting testability. Not many tactics were applied to address cybersecurity or data privacy. The reason for that is the fact that the architecture of the user interface module having an internet connection and user data storage is not within the scope of this thesis. Nonetheless, security related tactics *"prevent reverse engineering" and "protect interfaces"* were applied where applicable.

The view template was not applied to the *"+1"* view, the *"Scenarios"*. The purpose of the scenarios is to combine all other views with scenarios or use cases that describe how the components in the system interact with each other's in different situations. As an example, Figure 21 below presents a set of use cases for a software item of the software architecture of the company project.

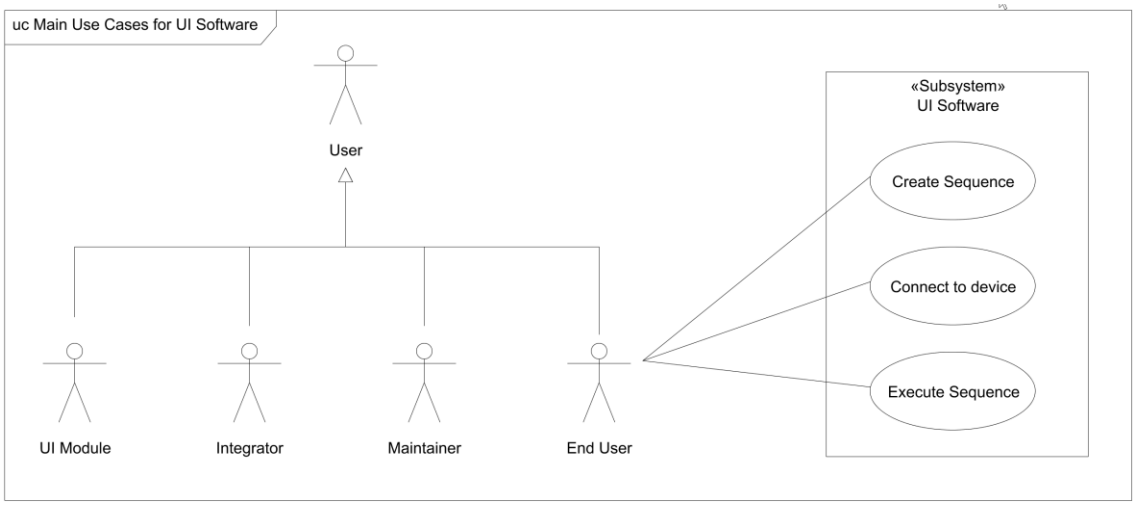


Figure 21. UI software use cases. Notation: UML use case diagram.

The Figure 21 is a UML use case diagram, in which a type of user is an actor in a several use cases. The user inheritance part of the diagram is for illustrating that all the different user types are *"Users"*. For each use case, there is a UML sequence diagram, which is used to demonstrate how different software items interact with each other's in the use cases in question. Sequence diagram is a variant of UML interaction diagram. As an example, the sequence diagram related to the *"Connect to device"* use case is presented in the Figure 22.

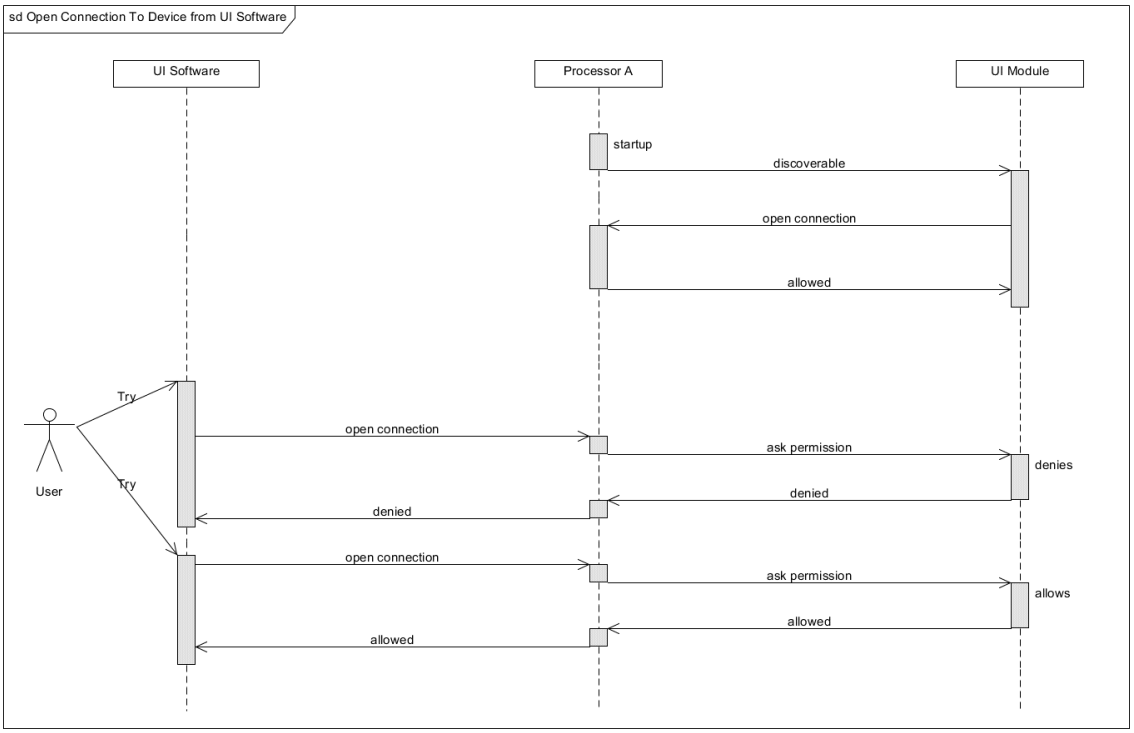


Figure 22. Interactions of “Open Connection” use case. Notation: UML sequence diagram.

The sequence diagram in the Figure 22 presents lifelines of three separate software items and a user. The arrows in the diagram represent messages between the software items. The boxes illustrate activation or active processing in a software item. The software item in the middle accepts connections from software items both left and right. In the history of the sequence diagram before a user action, a connection is established with the software item on the right. The user using the software item on the left is forced to retry a connection before the software item on the right allows it.

In the company project, the scenarios have been used for evaluating the architecture. The evaluations are performed in peer or stakeholder reviews. In a review, a certain aspect of the architecture is inspected by designing a scenario for it. The outcome of such session is that the scenario either is or is not possible with the design. If not, then the architecture have been revisited. The scenarios package is part of the software architecture model of the company project and is documented in the software design description of the project.

### 4.3.2 Interfaces between Software Items

The second requirement for software architectural design in the IEC 62304 standard is the interface architecture between software items and between software items and external software and hardware components. The standard does not explain the requirement any further, it plainly requires that the interface architecture must be developed and documented if the software safety classification of the software system in question is class B or C [5, cp. 5.3.2].

The interface architecture requirement of IEC 62304 is addressed in the views "*Functional Description"* and *"Task Architecture"* of the software architecture of the company project. The views are described in the Appendix 2. The view *"Functional Description"* thoroughly describes all interfaces between different computational units internal and external to the device. The description of interfaces between different tasks inside software items is visible in the view *"Task Architecture"*.

### 4.3.3 Functional and Performance Specifications of SOUP Items

IEC 62304 requires that software architecture must contain functional and performance specifications of SOUP items [5, cp. 5.3.3]. SOUP items are software of unknown provenance, or in other words, third party software. Outside the architecture requirements, the IEC 62304 requires also that SOUP items need to be identified. Information such as vendors and version numbers have to be documented [5, cp. 8.1.2].

The SOUP specifications are required for software systems having a software safety classification class B or C [5, cp 5.3.3 - 5.3.4]. The level of detail required for a SOUP specification is a subject that is not very clearly addressed in the IEC 62304. For example, the standard expects that a manufacturer of a medical device must identify standard libraries as SOUP items [5, cp. 8.1.2]. Constituting full functional and performance specifications for example of an open source C programming language standard library can be an overwhelming task. One approach to avoid specifying such library is to ensure that any software system using the library is classified to class A. Then only the SOUP identification is required. Another approach, with safety class B or C, is to use only commercial SOUP items from vendors who are able to provide the required specifications.

The company project related to the thesis indeed contains third party software items that would have to be specified. Examples of such are open source libraries and commercial file systems. The possible medical intention and software safety classifications of the software systems of the company product under development were not within the scope of this thesis and thus the SOUPs are not specified but only identified using the template presented in the Appendix 3. The identified SOUPs are listed in the software design description of the company project.

### 4.3.4 Hardware and Software System Specifications of SOUP items

Computational environment requirements of SOUP items are obligatory for software safety classes B and C. A note in the requirement suggests that the system specifications could include topics such as *"processor type and speed, memory type and size, system software type, communication and display software requirements"* [5, cp. 5.3.4]. The specifications are needed when a software architecture of a software system containing SOUPs is verified.

The possible medical intention and software safety classifications of the software systems of the company product under development were not within the scope of this thesis and thus the environmental specifications of the SOUPs are not formally specified.

### 4.3.5 Segregation Necessary for Risk Control

By segregation, the IEC 62304 means separation of parts of the system effectively enough that they are not *"negatively affecting"* each other's. Effective segregation allows software items to be classified in different software safety classifications [5, cp. B.4.3]. Software items running on separate computational units without shared resources is an example of a segregation. Identifying of segregation is required only for software systems with software safety class C [5, cp. 5.3.5]. .

Various software items of the device under development in the company project are allocated to separate computational units. The functions related to reliable operation and performance are allocated to the main control processor of the device. A co-processor supporting the main controller is responsible for less critical functions. A separate internal UI module provides planning and light interaction functionality. An UI software running on an external computer has similar features than the UI module. Least critical

functionality, including monitoring and such, is implemented in cloud services. In the hypothetical case where the device would have a medical intention, the risk levels and software safety requirements would increase in the reverse order compared to what was listed above. The details of the segregation in the software system of the company project is presented in the architectural view *"Executive View"* of the Appendix 2.

### 4.3.6 Software Architecture Verification

As presented in the background chapter 2.5, IEC 62304 architecture verification requirements can be met by conducting a technical evaluation [5, cp. B.5.3]. In the company project, the evaluation is done in peer reviews with participants from roles such as system architects, software architects, technical product owners and lead developers. Reviews have been organized during the project whenever it has been necessary to assess if a certain part of the architecture is suitable for its purpose. The overall software system architecture has been reviewed in the very beginning of the design phase. Use cases and scenarios has been created in the reviews for evaluating the interface architecture.

The behavioral architecture designed using UML state machine diagrams is verifiable both in a simulated environment and in reality. This is possible by utilizing supporting features of the state machine framework in use. First, state transitions can be triggered by injecting events to the system. Secondly, each transition can be logged to the external interfaces of the system. Generally, the current state of any machine can be queried from the interfaces at any given time. In addition, the IDE in which the simulation of the system is developed contains a test framework that supports unit testing. The features described above enable testing of the structures of every machine by walking through every state transition in unit tests. In practice, the dynamic behavior of the machines may create some difficulties in testing and writing the tests requires considerable effort, but theoretically unit testing can be used to cover the whole modelled behavior of the simulated system. Furthermore, the unit tests can be used to verify both external and internal interfaces using event injection in the simulated environment.

The monitoring and event injection capabilities of the state machine framework are in use also in the real target environment. This is useful in development phase for troubleshooting but especially useful when verifying the architecture in integration and system testing with real devices. An example of an event injection in system testing could

be a scenario where an error event is injected to the application state machine. Referring to *"Processor A"* behavior description in the view *"Functional Description"*, the application should make a transition to the state *"Error"* regardless in which sub state of state *"Run"* the application is. The test case would record the current state, inject the error event and verify the state of the application after the injection.

IEC 62304 requirements for architecture verification of software with safety classifications B and C are traceability of risk control related requirements to architecture, reasonable interface architecture and support of SOUP items [5, cp. 5.3.6]. The software architecture description is a part of the software design description of the product under development in the company project. The software life cycle management tool of the company is capable of producing traceability matrixes between requirements and other items that it manages, such as meeting minutes of reviews and changes in the software design description. The interface architecture evaluation is part of the technical reviews. The SOUP items are identified but not specified in the company project, as explained in the chapter 4.3.3. If the device under development would have a medical intended use and a software safety classification higher than class A, then the technical evaluations would have to address the SOUPs also. However, the SOUP evaluations were not in the scope of this thesis.

## 4.4 Interviews

This chapter presents the results of the interviews that were carried out for the purpose of evaluation this thesis. The evaluation was qualitative, although part of the interview questions produced quantitative answers. The questions used in the interviews followed the interview form in Appendix 1. Three interviews were conducted. The interviewees were well aware of the study under evaluation and had read the thesis before the interviews. Not all chapters of the thesis were finished before the interviews, including this chapter and chapters *"Discussions and Conclusions"*, *"Summary"* and *"Abstract"*. The interviews were recorded to audio/video, from which the answers were transduced to subjective raw data. The raw data of the interviews is available in Appendix 4. Following are the persons interviewed.

- *Interviewee #1:* Senior Project Manager
- *Interviewee #2:* Senior R&D Manager
- *Interviewee #3:* Workstation Software Architect

Interviewee #1 represented stakeholders from project management and product management and was the project manager of the company project related to this thesis. The management team of the product included members from business and marketing, thus the voice of customers was represented indirectly. On the other hand, for the development team, the customer is the product management. Interviewee #2 is a software manager in the company, responsible of software development, verification, processes and standard operating procedures (SOPs). In addition, interviewee #2 represented the quality department of the company via the SOP relation. Interviewee #3 is a peer software architect of the interviewer in the company, representing system designers, software designers and software developers. Interviewee #3 participates in the development of the product related to this thesis. The responsibilities of the interviewee #3 were on designing higher-level software items, meaning software items related to user interfaces and experience. The interviewees are addressed with their titles from hereafter.

The interviewees had different expectations or concerns towards the study. The project manager was expecting that the study would increase company knowledge on details of IVD medical device product development. The software manager had concerns on all the standards and regulations of the field in which the company operates and was expecting that the study would answer to the concerns from the viewpoint of software development, thus easing the forthcoming task of updating the SOPs of the company. In the beginning of the study, the architect expected that the thesis would generally clarify how the standards and regulations need to be taken in to consideration in software development. Later, the architect was positively surprised that the work was also related to the same product development project the architect participated.

The overall impressions of the interviewees on the thesis were good. The project manager thought that the thesis had succeeded in presenting the topic of software architecture development essentials in the regulated environment in question. The software manager agreed that the software standards and the difference of IVD and non-IVD product development were covered. The architect thought that especially the structure of the thesis is logically organized to background, theory and practice. However, the text was considered as heavy reading by all of the interviewees.

The interviewees were asked to grade the level of significance of the study in a scale of low–medium–high. The results are summarized in the Table 2.

Table 2. Level of significance of the thesis.

| Level of Significance | | | |
|---|---|---|---|
| | **Low** | **Medium** | **High** |
| **Interviewee #1:** | | | **X** |
| **Interviewee #2:** | | | **X** |
| **Interviewee #3:** | | | **X** |

The interviewees all graded the level of significance of the thesis high, as is visible in the Table 1. Both project and software manager thought that the study had created knowledge, which can be relied on when justifying decisions in product development projects. The architect commented that the thesis is valuable for any software architect or developer who needs to increase knowledge on the subject of the thesis.

The goals of the thesis were understood reasonably well. The project manager knew that clarifying the topics related to the regulated environment and analyzing the SOPs of the company were among the goals. The software manager and the architect understood also that the regulated environment was studied from the perspective of the software architecture design of the ongoing company project. All interviewees thought that the goals were achieved, at least after the goals were reminded to them. The project manager commented that the thesis had managed to delimit a very large topic to a reasonable size. The software manager considered the realization of the fact, that architectural decisions based on sufficient knowledge of the thesis topic are important in the beginning of the design phase, as an achievement.

The background for the subject presented in the thesis captured the essentials, commented the project manager. The software manager felt a need to revisit the background, for ensuring that the software manager had obtained all the necessary information on standards and regulations required to be followed by the company. The architect thought that a wider handling of the subject would not had improved the understanding anymore, there is so much information in the theory part already. In addition, the software manager commented that the background section is massive.

Nevertheless, the structure of the thesis was considered good by all the interviewees. In addition, the architect commented that the background–theory–practice structure is clearly visible in the thesis. The software managers' opinion was that even though the thesis addresses so many topics, no sections could be left out.

All interviewees recognized the thesis as useful. The fact that the company confidential information is not presented in the thesis was seen as a positive thing. *"Public audience is able to benefit from the study also"*, said the project manager. In addition, the benefits for the company were highlighted. The project manager thought that the outcomes of the work are already visible in the architecture design of the ongoing project. The software manager emphasized the support that the thesis provides for SOP development. Both the software manager and the architect considered that the thesis is going to be a useful resource for them and others whenever there is a need to refresh knowledge on the subject.

The interviewees saw that the thesis creates opportunities for future development, mainly for SOP development. In addition, some ideas aroused in the interviews, such as analyzing of regulations of market areas outside EU more thoroughly and clarifying the requirements for software development in ISO 9001 quality system standard series. Lastly, the architect summarized the opportunities by commenting that in the future there should be less confusion around the topics related to the thesis.

# 5 Discussions and Conclusions

The goals of this thesis were set ambitiously to answer all the questions and needs that the company had at the time when the study started. Conveniently, there was a product development project starting, which required system and software architectural design contribution. Simultaneously, the changes in the regulated environment had raised many questions, some of which were given to the study to be solved. From this setup to goals of the thesis became analyzing the new regulated environment from the point of view of product development and especially software development, analyzing product development standard operation procedures (SOPs) of the company and recommending updates, clarifying the responsibilities of manufacturers supplying to IVD medical systems and designing and describing software system architecture following industry standards.

The review of the regulated environment focused on new European legislation and medical device product development life cycle standards. Some aspects of U.S. medical device legislation were addressed also. The focus of the narrative in the background chapter was intended to be kept on software development. However, there were not so many changes found directly related to software. Thus, the current state of medical device software development requirements was confirmed: software life cycle processes according to standard IEC 62304 and within a medical device quality management system compliant to standard ISO 13485 and not forgetting risk management requirements of standard ISO 14971. Nevertheless, changes outside the software context were summarized. The most important change for IVD medical device development is the new risk classification system. It will force all IVD products to be re-classified.

The foremost conclusion from the analysis of the SOPs of the company was the fact that there were no clear differentiation between guidance for IVD and non-IVD development. The recommendation for clarifying the difference together with other recommendations were summarized in results Table 1. The company recognizes two quality management system, which partially overlap. However, standard ISO 13485 is a based on ISO 9001 and IEC 62304 acknowledges ISO 90003. In that sense, the processes of the company for medical and non-medical products are not completely incompatible. Nevertheless, ISO 90001 pursues product quality and reliability, whereas ISO 13485 aims for patient and user safety. The biggest difference for software development is in the risk

management processes – IEC 62304 requires that medical device manufacturers have to consider how software failures might affect safety.

The question of responsibilities of manufacturers who supply components to medical devices was illuminated in the light of new European medical device and IVD medical device laws. It was found that the obligations lay on those who utilize the components in their systems. Exception to the rule is if a component manufacturer specifically intends a product to be used in a particular IVD-system as an IVD accessory. Otherwise, the component manufacturer should only be prepared to provide necessary performance and usage specifications for system manufacturers who need to address those, for instance, in their risk management and system verification processes. However, it was noted, that the responsibilities of the system manufacturers might include auditing the premises and processes of the component manufacturer. The aforementioned result was achieved by a plain study of the new medical and IVD medical device regulations. No evaluation were performed for the interpretation of the laws.

A software development SOP of the company requires that an architecture of a software has to be included in a design description of the software and that the architecture should be evaluated by reviews. No further guidance is given. The main goal of this thesis was to introduce how a reusable way of describing architectures can be used to fulfill that requirement in a manner that provides compliancy against industry standards and especially against standard IEC 62304. The work was performed by designing and describing a software system architecture for a product under development in the company. The product is an electromechanical instrument, which is controlled by embedded software. For that reason, also the background theory of the thesis focused mostly on software architectural methods and patterns suitable for embedded systems.

Many design decisions used in the software were known by the company from former similar products. The new thing was the way, how they were selected and described in the architecture before implementation. The architectural strategy was built on design principles written in the product requirements and architectural patterns were utilized first to software system architecture and then in detailed design. A software design is something in which an architecture exists, whether or not it is described. In addition, describing a complex architecture with a single view is impossible [20, cp. 1]. The solution to the problem is to provide multiple views, each of them describing the architecture for a specific viewpoint. The standards and papers in the field of software

architecture development very much emphasize this philosophy, as was demonstrated in the background chapter.

This thesis utilized a commonly known view model and a view description template for describing the software system architecture of the company project. The presentation format of the description in this thesis differs from the actual included in the software design description of the company project, which uses hypertext markup language. The selected view documentation template itself can be used in any documentation format. The publisher of the book from which the template was obtained provides the template as Wiki-pages [50]. There are other templates available online. One example is the *"System Architecture Document"* from software development process templates repository of a company specialized in medical device development consulting. Their template is especially targeted to address requirements of IEC 62304 and also FDA, GDPR and others [51].

A good part of the system behavior was modelled in the architecture using UML state machine diagrams and activity diagrams. Thus, that part of the architecture was driven by models. Model driven architecture (MDA) is a design method that moves the focus from software implementation to software design or architecture, creating visibility and preconceived reason to systems. Behavioral modelling using UML was not a new concept in the company. However, describing nearly everything else in the architecture with UML was new, including interfaces, module decomposition, processes, physical deployment and interactions between software items. In addition, an UML extension SysML was introduced in the architecture description. The diagrams were designed in an agile modelling tool and were exported to the design description of the project in a picture file format. A development opportunity for the company could be an evaluation of formal SysML/UML modelling environment. Ultimately, such tool could be used for automatic code generation, which would further enhance the paradigm shift from implementation centric software development to design centric development. Another benefit of utilizing SysML could be an automatic traceability analysis from software system requirements to software system architecture. A traceability analysis can be used to prove compliancy against an IEC 62304 software architecture verification requirement [5, cp 5.3.6].

Verification of the architecture in the company project was based on technical evaluation. In practice, the architecture was reviewed. Particularly useful were the sessions in which scenarios were designed in addition to plain reviewing. Meeting minutes of those reviews

where UML use case and sequence diagrams. Revisiting an architecture is something that is highlighted in the standard ISO 42010 *"Systems and software engineering — Architecture description"*. The architecture has a life cycle; it is not just a single shot design effort in the beginning of a project. For future development, the company could formalize the technical evaluation process, for instance, by providing a template for the meeting minutes of the reviews.

IEC 62304 requirements for software of unknown provenance (SOUP) were addressed superficially in the thesis. For an architecture of a software with safety class B or C, the requirements include performance, functional, software system (OS), and hardware specification and a requirement for verifying that the architecture supports correct operation of the SOUPs. Elsewhere in the standard, the SOUPs are mentioned in various other requirements including development planning, software integration, testing, risk management, configuration management, software requirements and maintenance. Some of the requirements concern software with safety class A also. The approach to the SOUP issue in the thesis was to identify the SOUPs, which is required for all safety classes. However, a much larger effort is required for classes B and C. For example, medical device manufacturers are obligated to follow published anomaly lists (bug reports) of the SOUPs and consider the potential risks that the formerly unknown anomalies might cause. For future development, the company could improve the software SOPs or work instructions around these topics. [5]

The relevance of this thesis was evaluated by interviewing a group of people from the company whose product development project the study was related to. Without exceptions, the interviewees found the level of significance of the thesis to be high. A conclusion from this is that there was a need for the study in the company. In addition, the interviewees considered the work relevant outside the company also. However, evaluating that was not in the scope of the thesis.

# 6 Summary

The target of this thesis was to research how software system architectures are developed under medical device regulations while designing a one in a product development project of a company. The explicit objectives of the study were the analysis of the product development and software development standard operation procedures (SOPs) of the company and recommendations for updates; researching of the responsibilities of manufacturers that supply components to in vitro diagnostic (IVD) medical systems; and designing and describing a software system architecture in a company project.

The theory section of this thesis presented the regulated environment in which the company operates. The study concentrated on the new laws EU 2017/745 for medical devices and EU 2017/746 for IVD medical devices. In addition, general data protection regulation and parts of the medical device regulations set by food and drug administration of United States of America were addressed. Concept of harmonised standards was introduced: a list of specific standards, which are recommended to be followed for a compliancy against a specific law. The most relevant ones for medical device software development in Europe are ISO 13485 "*Medical devices — Quality management systems — Requirements for regulatory purposes"* and IEC 62304 *"Medical device software - Software life cycle processes"*. Aforementioned and other related standards were presented from the viewpoint of software development. In addition, requirements for architecture development in the said environment and generic concepts of embedded software system architectures were discussed in detail.

The methods of the study were literature reviews and gap analyses of related regulations and standards; analysis of company material by reflecting them on the regulations and standards; and describing architecture of the company product by utilizing views and viewpoints philosophy, view documentation template and unified modelling language (UML) diagrams. The method for evaluating the study was interviewing.

The analysis of the company SOPs produced eleven recommendations for improvements, from which a recommendation for clarifying the differences between requirements of IVD and non-IVD development was the most relevant one. The responsibility for appropriateness of components in IVD systems was found to be in practice with system manufacturers. Only in the case of an accessory for an IVD medical device a component manufacturer is under legislation of the IVD medical device

regulation. The software system architecture description presented five different views utilizing several different type of UML diagrams. Architectural design decisions and patterns were documented and utilized. The design in the company project was a combination of model driven and event driven architectures with a modular structure. In addition, the study addressed also all the requirements set by IEC 62304 for software architectural design of medical device software.

The study was closed with a qualitative evaluation of the results. It was performed by interviewing a selected group of stakeholders of the company project and the thesis. In overall, the interviewees had a good impression of the study. Moreover, each one of the interviewees found the study to be highly significant.

## References


[1] The European Parliament and the Council of the European Union, "REGULATION (EU) 2017/745 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EE," *Official Journal,* no. L117, 2017.

[2] The European Parliament and the Council of the European Union, "REGULATION (EU) 2017/746 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 5 April 2017 on in vitro diagnostic medical devices and repealing Directive 98/79/EC and Commission Decision 2010/227/EU," *Official Journal,* no. L117, 2017.

[3] International Organization for Standardization, *International Standard ISO 13485 Medical devices — Quality management systems — Requirements for regulatory purposes, third edition, reference number ISO 13485:2016(E),* ISO, 2016.

[4] European Commission, "Harmonised Standards," [Online]. Available: http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards. [Accessed 20 January 2018].

[5] The International Electrotechnical Commission, *IEC 62304 Medical device software - Software life cycle processes, Edition 1.1.,* IEC, 2015.

[6] U.S. Food and Drug Administration, *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices,* U.S. Department of Health and Human Services, May 11, 2005.

[7] H. Gomaa, Real-Time Software Design for Embedded Systems, Cambridge University Press, 2016.

[8] FOOD AND DRUG ADMINISTRATION, DEPARTMENT OF HEALTH AND HUMAN SERVICES, "Title 21, Chapter I, Subchapter H, Part 809 IN VITRO DIAGNOSTIC PRODUCTS FOR HUMAN USE, Subpart A—General Provisions, §809.3 Definitions," U.S. Government Publishing Office, [Online]. Available:

https://www.ecfr.gov/cgi-bin/text-idx?SID=ad8f516420ce59c30dd5a76051544875&mc=true&node=pt21.8.809&rgn=div5#sp21.8.809.a. [Accessed 23 February 2018].

[9] U.S. Food and Drug Administration, "Is The Product A Medical Device?," [Online]. Available: https://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/Overview/ClassifyYourDevice/ucm051512.htm. [Accessed 23 February 2018].

[10] U.S. Food and Drug Administration, "Code of Federal Regulations, Title 21--Food and Drugs, Chapter I--Food and Drug Administration H--Medical Devices, Part 820 Quality System Regulation," U.S. Government Publishing Office, [Online]. Available: https://www.ecfr.gov/cgi-bin/text-idx?SID=3fc1b63a4d173d7792fc5aaa6c2b8078&mc=true&node=pt21.8.820&rgn=div5#se21.8.820_130. [Accessed 4 March 2018].

[11] European Commission, "Harmonised Standards - Medical devices," [Online]. Available: https://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/medical-devices_en. [Accessed 4 March 2018].

[12] European Commission, "Harmonised Standards - In vitro diagnostic medical devices," [Online]. Available: https://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/iv-diagnostic-medical-devices_en. [Accessed 4 March 2018].

[13] U.S. Food and Drug Administration, "Recognized Consensus Standards," [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfStandards/search.cfm. [Accessed 24 July 2018].

[14] U.S. Food and Drug Administration, "Fact Sheet: FDA Good Guidance Practices," [Online]. Available: https://www.fda.gov/AboutFDA/Transparency/TransparencyInitiative/ucm285282.htm. [Accessed 24 July 2018].

[15] European Comission, "Medical Devices - Guidance," [Online]. Available: https://ec.europa.eu/growth/sectors/medical-devices/guidance_en. [Accessed 24 July 2018].

[16] U.S. Food and Drug Administration, "Global UDI Database (GUDID)," [Online]. Available: https://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/UniqueDeviceIdentification/GlobalUDIDatabaseGUDID/default.htm. [Accessed 26 July 2018].

[17] European Commission, *Factsheet for Manufacturers of Medical Devices, ISBN: 978-92-79-89634-7,* European Union, 2018.

[18] European Commission, *Factsheet for Manufacturers of In-Vitro Diagnostic Medical Devices, ISBN: 978-92-79-89707-8,* European Union, 2018.

[19] T. E. Bell and T. A. Thayer, *Software requirements: Are they really a problem? Proceedings of the 2nd international conference on Software engineering.,* IEEE Computer Society Press, 1976.

[20] Bass, Clements and Kazman, Software Architecture in Practice (3rd Edition), Addison-Wesley Professional , 2012.

[21] U.S. Food & Drug Administration, "Recognized Consensus Standards - ISO 14971 Supplementary Information Sheet," [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfStandards/detail.cfm?standard__identification_no=37014. [Accessed 13 November 2018].

[22] The International Electrotechnical Commission, *IEC 60601-1:2005 Medical electrical equipment - Part 1: General requirements for basic safety and essential performance,* IEC, 2005.

[23] The International Electrotechnical Commission, *IEC 62366-1:2015, Medical devices - Part 1: Application of usability engineering to medical devices,* IEC, 2015.

[24] U.S. Food & Drug Administration , "Recognized Consensus Standards - IEC 62366-1 Supplementary Information Sheet," [Online]. Available:

https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfStandards/detail.cfm?standard__identification_no=37018. [Accessed 13 November 2018].

[25] International Organization for Standardization, "Standards catalogue," [Online]. Available: https://www.iso.org/standards-catalogue/browse-by-ics.html. [Accessed 28 July 2018].

[26] U.S. Food & Drug Administration , "Recognized Consensus Standards - IEC 62304 Supplementary Information Sheet," [Online]. Available: https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfStandards/detail.cfm?standard__identification_no=33542. [Accessed 13 November 2018].

[27] U.S. Food and Drug Administration, *Content of Premarket Submissions for Management of Cybersecurity in Medical Devices,* FDA, 2014.

[28] European Union Agency For Network And Information Security, *Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures, ISBN: 978-92-9204-236-3,* ENISA, 2017.

[29] The European Parliament and the Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Da," *Official Journal,* no. L119, 2016.

[30] I. Sommerville, Software Engineering (9th ed.), Addison-Wesley, 2010.

[31] Institute of Electrical and Electronics Engineers, Inc., *International Standard 42010:2011(E), Systems and software engineering — Architecture description,* ISO/IEC/IEEE, 2011-12-01.

[32] P. Kruchten, "Architectural Blueprints — The “4+1” View Model of Software Architecture," *IEEE Software,* 42-50, November 1995.

[33] E. Gat, "On Three-Layer Architectures," in *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, AAAI Press, 1998, 195-210.

[34] AUTOSAR, "Layered Software Architecture (release 4.3.1)," 8 December 2017. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf. [Accessed 3 March 2018].

[35] C. Hobbs, Embedded Software Development for Safety Critical Systems, CRC Press, 2016.

[36] B. M. Michelson, *Event-Driven Architecture Overview,* Patricia Seybold Group, February 2, 2006.

[37] D. Harel, “Statecharts: a visual formalism for complex systems,” *Science of Computer Programming Volume 8, Issue 3,* 231-274, June 1987.

[38] M. Kaufmann, P. Mishra and N. Dutt, Processor Description Languages, ISBN: 978-0-12-374287-2, Elsevier, 2008.

[39] Object Management Group, “OMG Unified Modeling Language TM (OMG UML) Version 2.5,” [Online]. Available: http://www.omg.org/spec/UML/2.5. [Accessed 2 August 2018].

[40] Object Management Group, "OMG Systems Modeling Language Version 1.5," [Online]. Available: https://www.omg.org/spec/SysML/1.5. [Accessed 11 August 2018].

[41] P. Lago, I. Malavolta, H. Muccini, P. Pelliccione and A. Tang, "The Road Ahead for Architectural Languages," *IEEE Software,* vol. 32, no. 1, 98 - 105, 2015.

[42] P. Eeles, *Describing Software Architectures with UML 2.0, IBM European Rational Technical Conference, October 17–21 2005,* Düsseldorf: IBM, 2005.

[43] International Organization for Standardization, “ISO 9000 family - Quality management - ISO 9001:2015,” [Online]. Available: https://www.iso.org/files/live/sites/isoorg/files/standards/docs/en/iso_9001.pptx. [Accessed 26 August 2018].

[44] European Commission, *MEDDEV 2.14/1 revision 2 IVD Medical Device Borderline and Classification issues,* European Commission, 2012.

[45] European Commission, *Manual on borderline and classification in the community regulatory framework for medical devices, version 1.18,* European Commission, 2017.

[46] Europen Commission, *MEDDEV. 2.14/2 rev.1 IVD guidance : Research Use Only products,* Europen Commission, 2004.

[47] U.S. Food and Drug Administration, *Medical Device Accessories – Describing Accessories and Classification Pathways,* U.S. Department of Health and Human Services, 2017.

[48] International Organization for Standardization, *INTERNATIONAL STANDARD ISO 14971:2007 Medical Devices, Application of risk Management to Medical Devices,* ISO, 2007.

[49] The International Electrotechnical Commission, *IEC 12207:2008 Systems and software engineering — Software life cycle processes,* IEC, 2008.

[50] Software Engineering Institute , "Template:ArchitectureViewTemplate," [Online]. Available: https://wiki.sei.cmu.edu/sad/index.php/Template:ArchitectureViewTemplate. [Accessed 27 October 2018].

[51] MD101, "Templates Repository for Software Development Process - System Architecture Document," [Online]. Available: https://blog.cm-dm.com/pages/Software-Development-Process-templates. [Accessed 27 October 2018].

## Appendix 1. Interview Form

| Interview Form | |
|---|---|
| **Interview #** | |
| **Title of the interviewee** | |
| **Introductive Questions** | |
| 1.1 | What is your overall impression of the study? |
| 1.2 | What is your relation to the project/thesis? |
| 1.3 | What expectations did you have towards the study and/or the architecture? |
| **Detailed Questions** | |
| 2.1 | What is the level of significance (low, medium or high) of the study? Explain. |
| 2.2 | Was the goal of the study clear? Was it achieved? |
| 2.3 | Is the background information comprehensive enough? |
| 2.4 | Is the structure of the study clear? |
| 2.5 | What is the usefulness of the study for you or your organization? |
| 2.6 | What kind of opportunities does the study create for future development? |
| Notes: | |

# Appendix 2. Company Project Software System Architecture Views

This appendix presents the views of a software system architecture of a software system of a product of the company.

## A2.1 Executive View

The *"Executive View"* provides an executive summary of the architecture. The usage of the view is a system overview for all stakeholders of the architecture, including the ones not having knowledge on software technology.

### A2.1.1 Overall Viewpoint

Figure A2.1 presents an *"Overall"* viewpoint to the view *"Executive View".* A stakeholder of the viewpoint is concerned if the product is going to be realized by the manner it was intended by product marketing or other party representing a customer. In order to conform to the viewpoint, the view describes how the overall system is constructed. The documentation convention for the view is a free-form one-slide presentation.

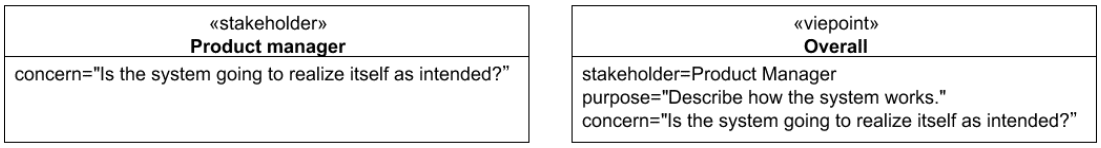


Figure A2.1. Viewpoint "Overall" to the view "Executive View". Notation: SysML.

### A2.1.2 Primary Presentation

The primary presentation of the view *"Executive View"* is presented in the Figure A2.2. Notation for the presentation is informal.

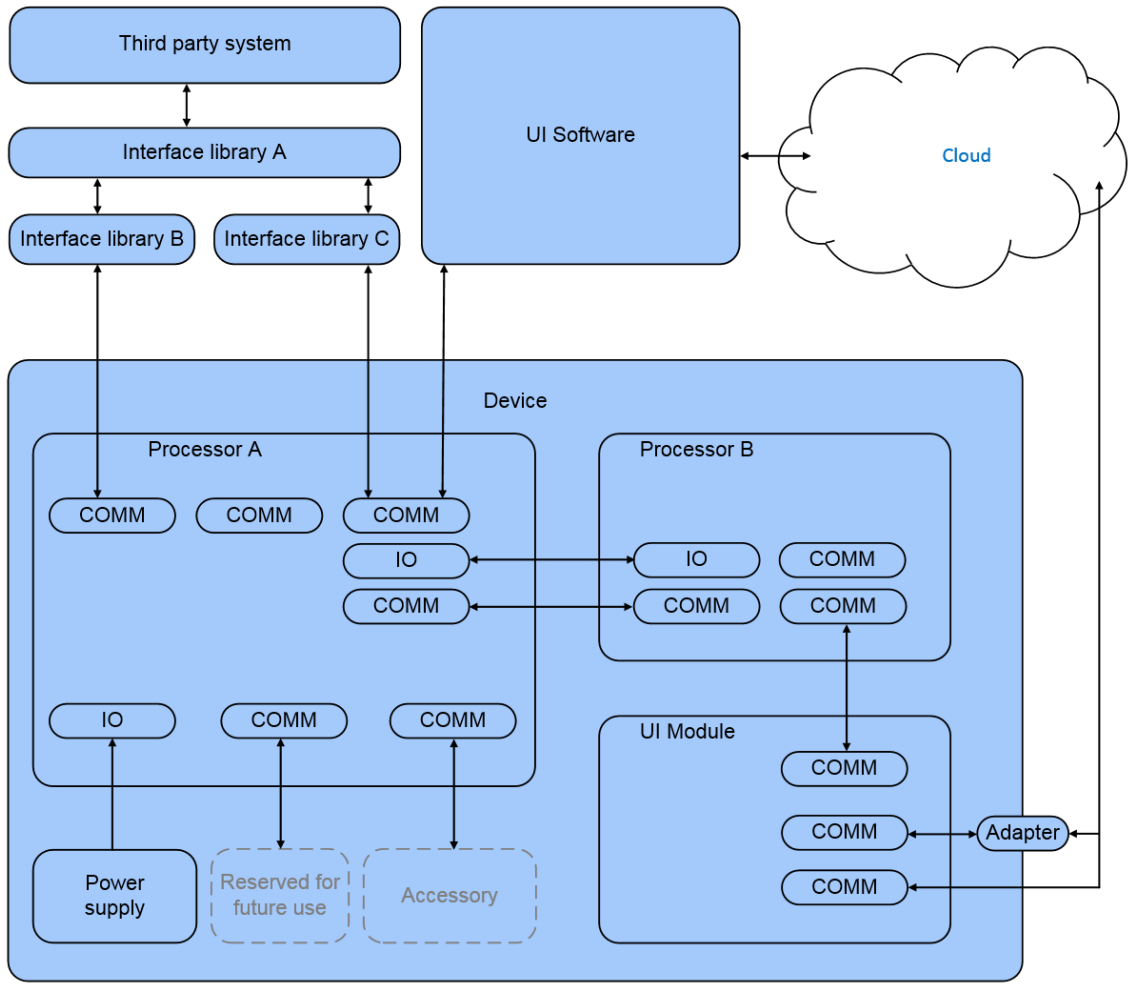


Figure A2.2. Executive summary level system diagram. Notation: Informal.

### A2.1.3 Element Catalog

This element catalog describes the elements presented in the primary presentation of the view *"Executive View"*

#### *A2.1.3.1 Device*

The element *"Device"* is the electromechanical entirety with the functionality specified in the design inputs of the project.

*A2.1.3.2 Processor A*

The element *"Processor A"* executes the software item that is mainly responsible of the device functionality, excluding the user interface and some supporting functionality provided by the software item executed in the element *"Processor B"*.

*A2.1.3.3 Processor B*

The element *"Processor B"* is a communication (COMM) bridge between the element *"Processor A"* and the user interface module. The interconnections between the processors A and B are shown in the Figure A2.3.

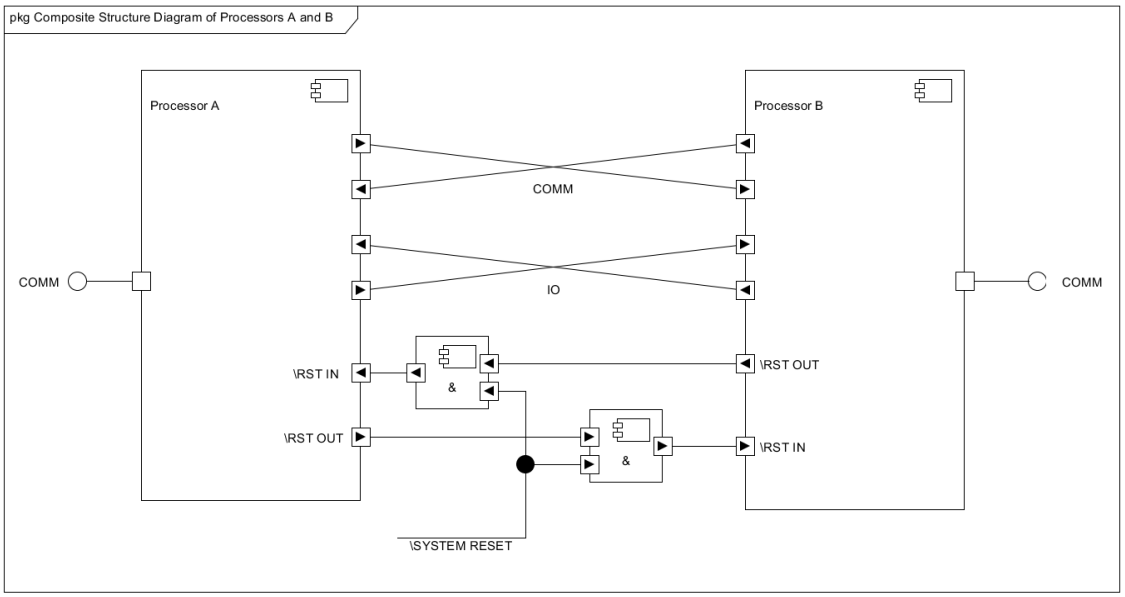


Figure A2.3. Interconnections of processors A and B. Notation: UML Composite Structure Diagram.

In order to achieve enhanced availability, the processors A and B are designed to be able to recover and upgrade the software items of each other's alternately. Both of them are able to reset the other and set it to a programming mode via input/output (IO) lines. The software item binary transfer is performed via the same COMM interface, which is used for normal communication.

*A2.1.3.4 IOs*

The IOs in the primary presentation are related to detecting power supply state and to interconnections between processors A and B. See elements *"Power Supply" and "Processor B"*.

*A2.1.3.5 COMMs*

Various communication ports in the primary presentation indicate bi-directional communication capabilities without specifying the actual communication standards/protocols or physical specifications of them. The COMM ports in the primary presentation without any connections are reserved for development, testing and troubleshooting purposes.

*A2.1.3.6 UI Module*

The user interface (UI) module is an execution environment for a user interface software item. The architecture of the UI software item is not within the scope of this architecture description.

*A2.1.3.7 UI Software*

The element *"UI Software"* is a similar or partly the same software item that is executed in the user interface module, see element *"UI Module"*. The UI software is executed in an external execution environment such as personal computer. The execution environment of the *"UI Software"* attaches to one of the COMM ports of the *"Device"*.

*A2.1.3.8 Cloud*

The element *"Cloud"* stands for specific internet online services that the UI software item utilizes in order to satisfy the requirements set for it and the element *"Device"*.

*A2.1.3.9 Adapter*

The element *"Adapter"* is an accessory device that can be connected to one of the COMM ports of the element *"UI module"*. The function of it is to transform the protocol in the COMM port in the manner that the UI software item can communicate with the element *"Cloud"* using the specific COMM port.

*A2.1.3.10 Third Party System*

The element *"Third Party System"* is and external computing environment that uses the services provided by the element *"Device"* through interface libraries provided by the company. The execution environment of the *"Third Party System"* attaches to one of the COMM ports of the *"Device"*.

*A2.1.3.11 Interface Libraries*

The interface libraries A, B and C are provided for third party systems for accessing the services of the element *"Device"*. Library A abstracts the COMM specific details and differences in the libraries B and C.

*A2.1.3.12 Power Supply*

The "Power Supply" element is capable of notifying the element *"Device"* about status of the input power.

*A2.1.3.13 Reservations and Accessories*

Part of the COMM ports of the element *"Device"* are reserved for unspecified accessories and other future expansions.

### A2.1.4 Context Diagram

The context diagram of the view *"Executive View"* is presented in the Figure A2.4. The purpose of the context diagram is to present the environment in which the device is in the center.

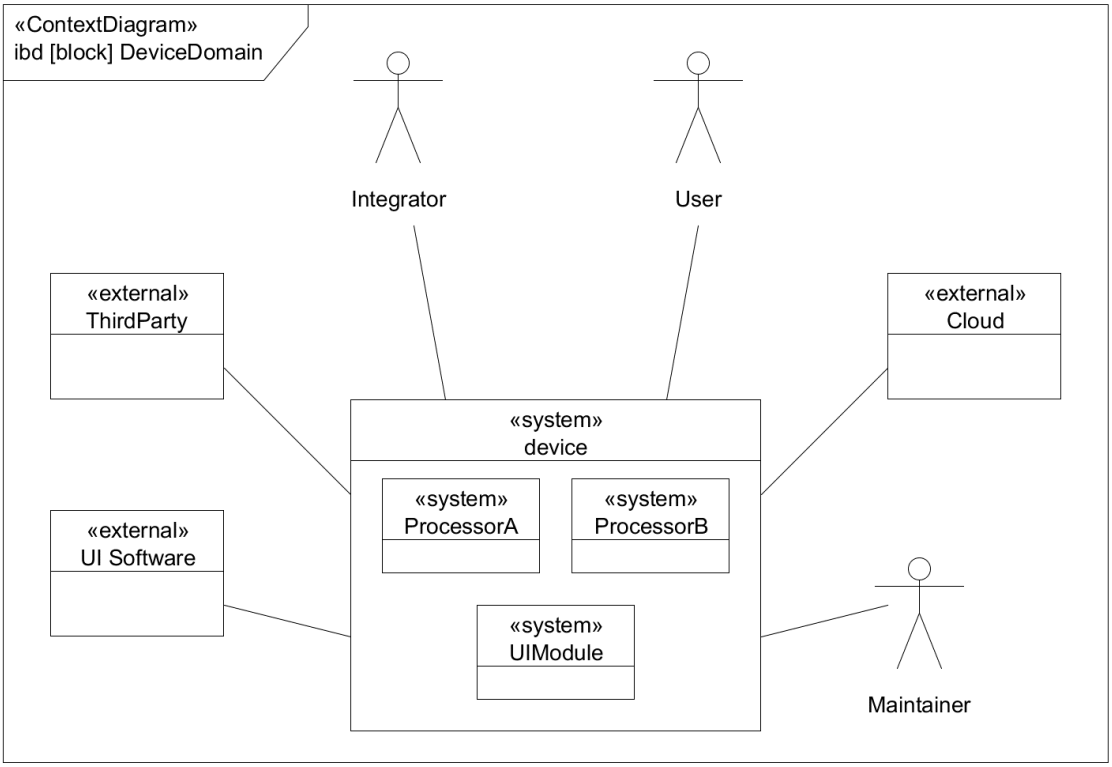


Figure A2.4. Context diagram of the view "Executive View". Notation: SysML internal block diagram.

The context diagram is a block diagram using user-defined stereotypes *"external"* and *"system".* The diagram is inspired by an example in the SysML specification [40, Fig. D.4]. In the diagram, the device is encompassed by the main parties of its environment. These include different type of human users and external computing environments.

### A2.1.5 Variability Guide

The types of every COMM port and the protocols used in them are actually specified in the project, but they are abstracted from this public version of the architecture description. Thus, there is no true variability in them. Accessories and future reservation systems connecting to the COMM ports reserved for them have to support the physical properties of the ports. Otherwise, the ports support variability on the types of the

connected systems and protocols. The third party system is not a specific one. Instead, it can be any system implementing the interface to the interface libraries.

### A2.1.6 Rationale

The dual processor device architecture was chosen to support availability and maintainability. Ability to recover from software item upgrade failure increases availability. Such failure might be caused, for example, by a loss of power during upgrading. It is especially important that the system can be upgraded from the cloud without fear of failure. Maintainability is elevated by the fact that the arrangement allows system to use in-build programming routines of the processors. In addition, the element *"Processor B"* has capacity for future expansions. Multi-tier pattern is used for segregating software items to different computational units.

## A2.2 Functional Description

The view *"Functional description"* describes the behavior and interfaces of the system. The view is relevant to those who need to know how the system functions and how the interfaces of the system are used. Interfaces and the behavior are described in detail in the software design description of the company project. This description presents a high-level view to that part of the SDD.

### A2.2.1 Logical Viewpoint

Figure A2.5 presents a *"Logical"* viewpoint to the view *"Functional Description"*. The primary concerns of the stakeholders of the viewpoint are how the system functions, how the system is used and what the behavior of the system is. Stakeholder for the concern is a user of the system. A user can be an end user, a customer, a maintainer, a third party system integrator or an external system. For the device, in the context of this thesis, the UI Module is an external user for the system.

| «stakeholder»<br>**User** |
| --- |
| concern="How does the system behave and how<br>are the interfaces of the system used?" |

| «viepoint»<br>**Logical** |
| --- |
| stakeholder=User<br>purpose="Describe how the system behaves and is used."<br>concern="How does the system behave and how<br>are the interfaces of the system used?" |

Figure A2.5. Viewpoint “Logical” to the view “Functional Description”. Notation: SysML.

### A2.2.2 Primary Presentation

The primary presentation of the view *“The* Functional *Description”* is presented in the Figure A2.6. Notation for the presentation is a UML component diagram. The presentation focuses on interfaces. Behaviour is addressed in the element catalog of the view.

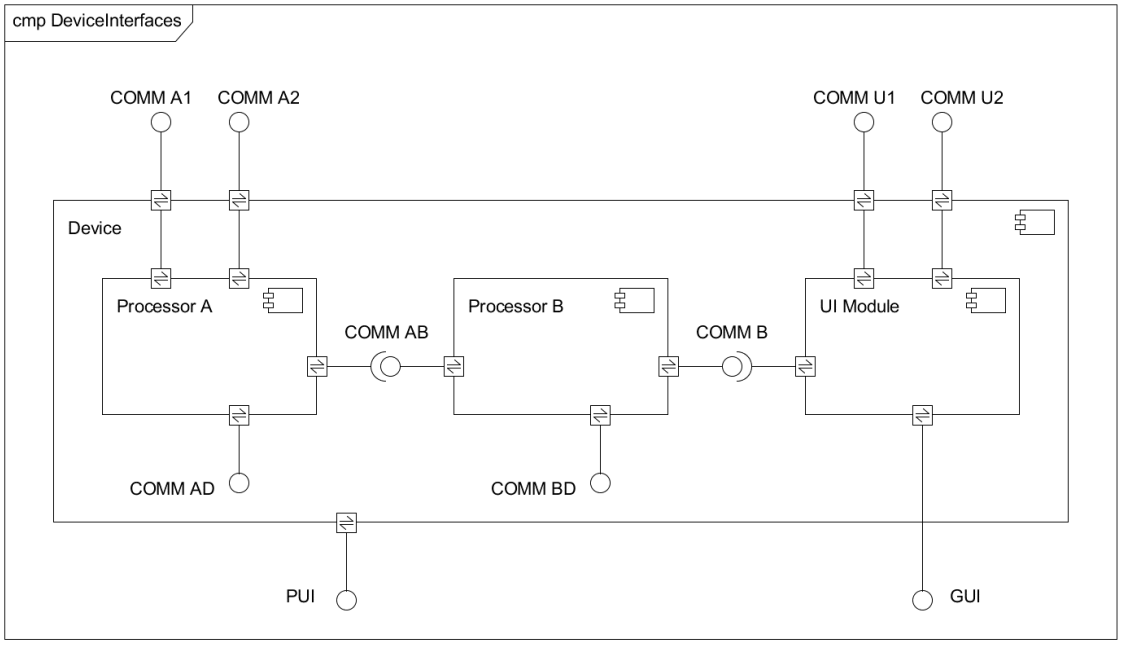


Figure A2.6. Interfaces of the device. Notation: UML component diagram.

The device provides several communication interfaces for external systems. Processor A and UI Module require interfaces from the Processor B. Both processors provide special interfaces, which are available only for development, problem solving and testing purposes. In addition to the machine interfaces, the device provides also a graphical user interface (GUI) and a physical user interface (PUI) for humans. The internal interfaces between software items in processors and in the UI module are not illustrated in the Figure A2.6 but are discussed in the element catalog.

### A2.2.3 Element Catalog

This element catalog describes the elements presented in the primary presentation of the view *"Functional Description"*.

#### *A2.2.3.1 COMM A1 and COMM A2 Interfaces*

The COMM interfaces A1 and A2 of the *"Processor A"* provide same logical interface A with different physical interfaces. The interface A provides access to such functionality as connection establishment, device control sequence transfers and executing of a sequence. The logical communication scheme is based on command-reply messaging with events. The user of the interface sends commands, which the *"Processor A"* replies. Events are asynchronous notifications to the user from the *"Processor A".*

#### *A2.2.3.2 COMM B Interface*

The *"COMM B"* interface includes logical interface A to the *"Processor A"* through *"Processor B"*. The interface B is required by the *"UI Module"* and it contains additional commands for upgrading the software item running in the *"Processor A"*. In addition, the interface B extends interface A with *"UI Module"* specific commands, replies and events. These include, for instance, events for notifying the *"UI module"* that a user is trying to establish a connection from interface *"COMM A1"* or *"COMM A2"*. For detecting a total failure of interface B, a tactic "send heartbeat" is used between the *"Processor A"* and the *"UI Module"*. The absence of such a periodic message is used as triggers for performing various recovery operations.

#### *A2.2.3.3 COMM AB Interface*

The *"COMM AB"* interface is provided by the *"Processor B"* for the "*Processor A*". The main purpose of the interface is to bi-directionally forward logical interface A messages between the *"Processor A"* and the *"UI Module"*. In addition, the interface includes commands for upgrading and configuring the software item running in the *"Processor B"*.

*A2.2.3.4 COMM AD and COMM BD Interfaces*

The D interfaces of the *"Processor A"* and the "*Processor B*" provide functionality that relates to monitoring and modifying the state and the data of the device. The users of the interfaces are software developers and testers.

*A2.2.3.5 COMM U1 and COMM U2 Interfaces*

The *"COMM U1"* and *"COMM U2"* are used for accessing local and wide area networks. These interfaces of the *"UI Module"* are not within the scope of this description. The functionality provided by the U interfaces are documented separately from the SDD referred by this document.

*A2.2.3.6 PUI and GUI Interfaces*

The physical interface of the device contains everything that a human user of the system can access physically. Actually, the graphical user interface is also accessed physically, but it is convenient to describe them separately. However, the GUI relates to the *"UI module"* and is not with the scope of this description. Furthermore, the PUI description relieves the true intended use of the device. For confidentiality reasons, it is not discussed in this description. In the end, the physical interface will be thoroughly described in the user manual of the product.

*A2.2.3.7 Device Behavior*

The behavior of the device refers to those physical events that the device performs due to different kinds of stimuli. A stimulus can be a human user action on PUI, GUI or on a graphical user interface of the *"UI Software"* running on an external computing system. Another type of stimulus is a state change in a software item controlling the actuators of the device via electro-mechanical link. The device behavior in this context is primarily described in the user manual of the product.

*A2.2.3.8 Processor A Behavior*

The behavior of the *"Processor A"* is a result of the behavior of the software item being executed in it. The software item consists of group of other items, each functioning independently. Each of those threads of a behavior are modelled into a form of a hierarchical state machine. The machines consume events, which in practice are messages in the internal communication bus of the underlying real-time kernel. Events cause processing and state transitions, in other words, behavior. The machines are modelled using UML state machine diagrams. The behavioral model of the main thread of the *"Processor A"* software item is presented in the Figure A2.7.

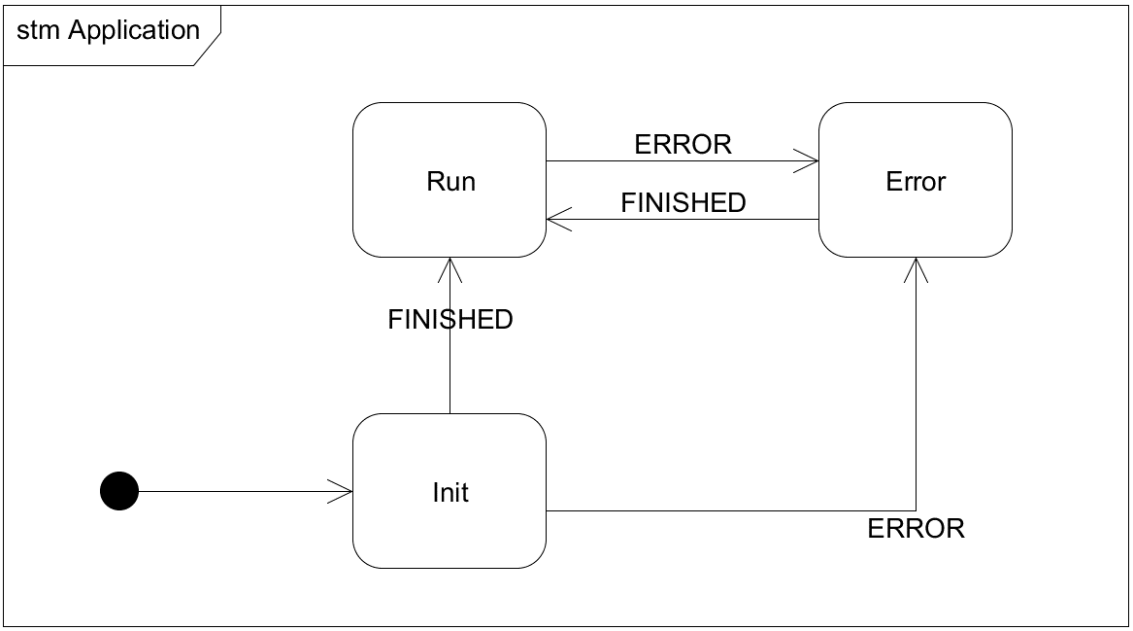


Figure A2.7. Application task state machine. Notation: UML state machine diagram.

The application state machine is ran by a task *"Application",* which is the main process of the *"Processor A"* software item. The application has three main states: *"Init"*, "*Run"* and *"Error"*. Each of the main states has a hierarchy of sub-states, state *"Run"* having the deepest. The error-handling pattern in use is eminently visible in the Figure A2.7. After the application is initialized, any occurring error causes a state transition to the state *"Error"* regardless of what is the current sub sub-state of the state *"Run"*. The same logic applies to the device initialization. Thus, all errors are handled in a uniform manner. The internal structure of the state *"Run"* is illustrated in the Figure A2.8.

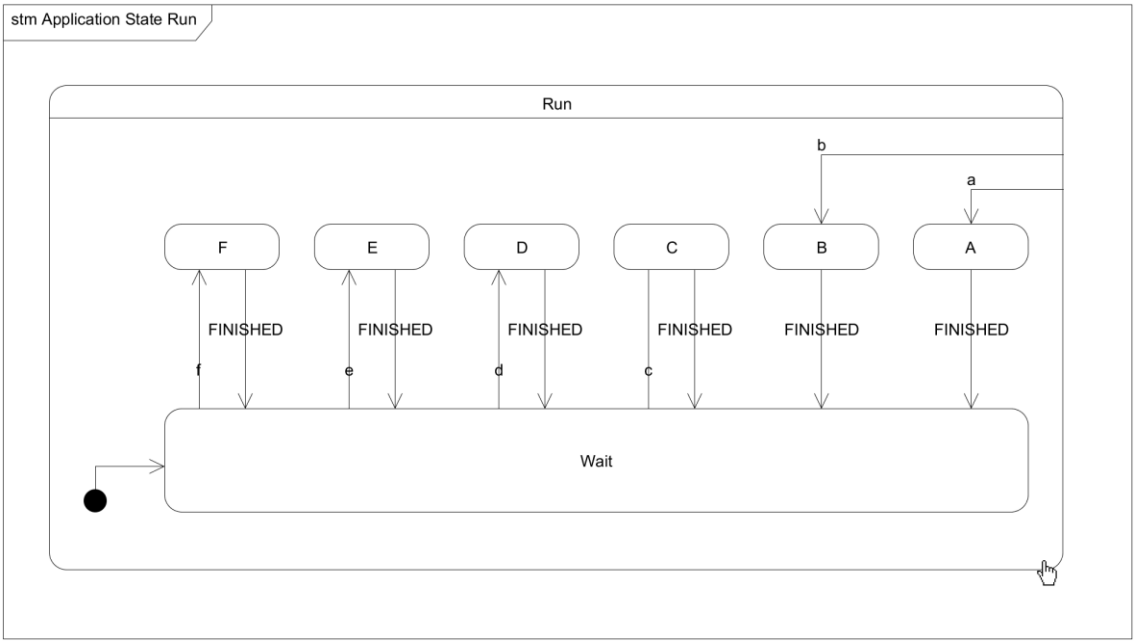


Figure A2.8. State “Run” of the application task state machine. Notation: UML state machine diagram.

The main concept of the application execution is to wait and execute commands coming from *“Processor A”* interfaces. The *“Run”* state has a sub-state a *“Wait”*, in which the application state machine waits those commands. A command is executed in one of the other sub-states, for instance in a state *“F”*. The *“Run”* state is designed in the way, that event *“a”* or *“b”* can cancel any command execution by causing a state transition to a specific state performing a specific function. An example of such function could be an emergency stop initiated by a user. The wholeness of the application state machine is described in detail in the software design description of the company project together with all the other state machine models of the various tasks of the *“Processor A”* software item.

### *A2.2.3.9 Processor B Behavior*

The software item functionality of the *“Processor B*” is not extensively complex and only one of the *“Processor B”* tasks is described using UML state machine diagrams. The task, named *“Application B”,* has the overall control to the *“Processor B”* behavior. Other tasks of the software item are supporting tasks and are mostly controlling communication and system services. The application state machine is presented in the Figure A2.9.

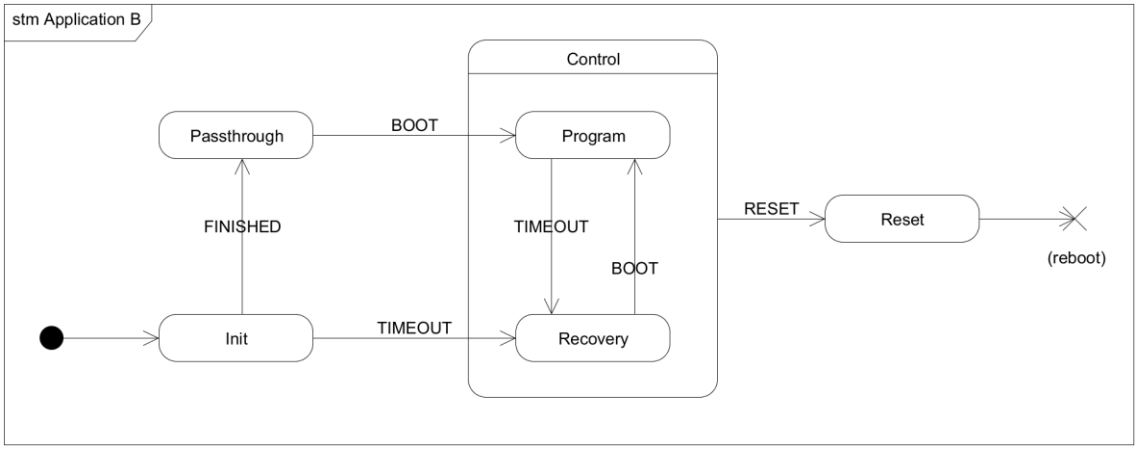


Figure A2.9. “Processor B” application task state machine. Notation: UML state machine diagram.

The *“Processor B”* application enters to an initialization state after startup. If the *“Processor A”* initializes the *“Processor B”* before a timeout event, then the application changes its state to a *“Pass-through”* state, in which all communication between interfaces *“COMM AB”* and *“COMM B”* is passed through the *“Processor B”*. In case of an initialization timeout, the application enters to a state *“Recovery”* instead. State *“Recovery”* is a sub-state of state *“Control”*, as is state *“Program”*. Transition to state *“Program”* is triggered by a *“Boot”* event, which is the one and only command that is detected from the *“COMM B”* interface in *“Pass-through”* state. The *“Reset”* finalizes programming by resetting the *“Processor A”,* which again, resets *“Processor B”* in successful start-up. For the case that *“Processor A”* would fail to start up the *“Processor B”* performs also a software reset to itself as a final action in the *“Processor B”* state. This ensures that the *“Processor B”* enters at least the state *“Recovery”* after exiting the machine. The above said describes the application B state machine in high-level, more detailed description of the machine is presented in the software design description of the company project.

#### *A2.2.3.10 UI Module Behavior*

The behavior of the *“UI Module”* is not within the scope of this description.

### A2.2.4 Context Diagram

The context diagram of the view *“Functional Description”* is presented in the Figure A2.10. In the figure, both *“Processor A”* and *“Processor B”* are placed in the middle of

their interfaces to emphasizes that these are the two main elements, whose software items are within the scope of this functional description.

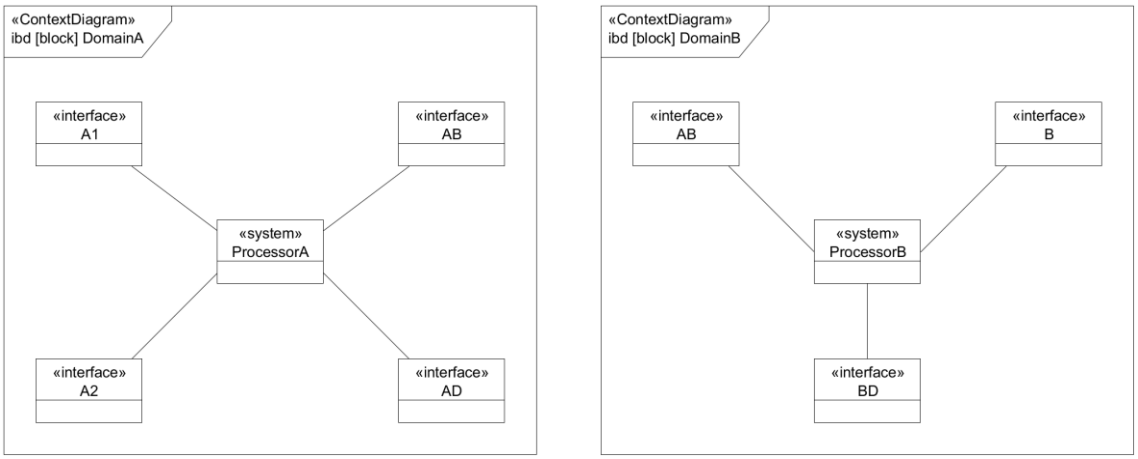


Figure A2.10. Context diagram of the view “Functional Description”. Notation: SysML internal block diagram.

### A2.2.5 Variability Guide

The behavioral models or the state machine diagrams have to be implemented as modelled. However, the designers are allowed to select the level of abstraction as long as the implementation of the modelled behavior can be verified from the real system. That is, what has been drawn must be true.

### A2.2.6 Rationale

The behavior and interfaces are designed to meet many of the quality attribute requirements that are set to the architecture. Modelling the software system behavior using UML state machine diagrams is the main pattern or tactic of the view. Detailed model supports maintainability and testability. Availability is taken into account by designing error handling in the core of the functionality. Testability is supported also by the monitoring interfaces. Description of the interfaces is a self-evident requirement for achieving proper interoperability. Protecting the interfaces is relevant to the *“UI Module”,* as it may be connected to information networks. The view uses also tactics *“provide debug interfaces”* and *“send heartbeats”.*

## A2.3 Development View

The view *"Development"* describes the structures of the software items that implement the architecture. The target audience of the description is the developers of the software system. The description in this document focuses on *"Processor A"* software item; other items are described in the software design description of the company project.

### A2.3.1 Development Viewpoint

Figure A2.11 presents a *"Development"* viewpoint to the view *"Development*. The main concerns of the stakeholders of the viewpoint are how and in what kind of environment the development of the software is performed. Stakeholders of the viewpoint include developers and testers. Stakeholders are also interested how the software modules are organized and what are the guidelines that need to be followed when implementing the software.

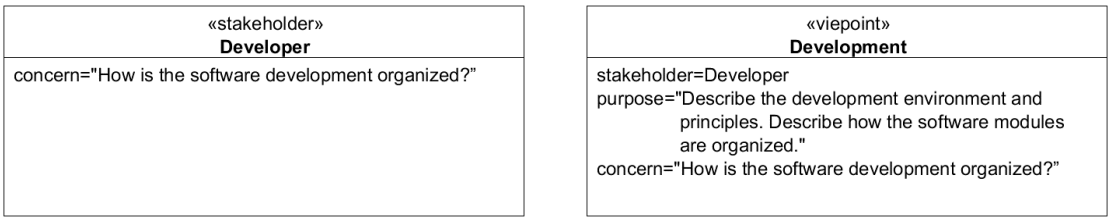


Figure A2.11. Viewpoint "Development" to the view "Development". Notation: SysML.

### A2.3.2 Primary Presentation

The primary presentation of the view *"Development"* presented in the Figure A2.12 focuses on software organization. Notation for the presentation is a UML package diagram. The dotted arrows show dependencies between the software items. The *"access"* stereotype emphasizes the direction of the dependency. The UML frame encloses the whole software item and the packages contain decomposed software items. When further decomposition is now longer practicable, then a package is called a *"software unit"*. The naming convention is compatible with the standard IEC 62304 [5, cp. 3.25].

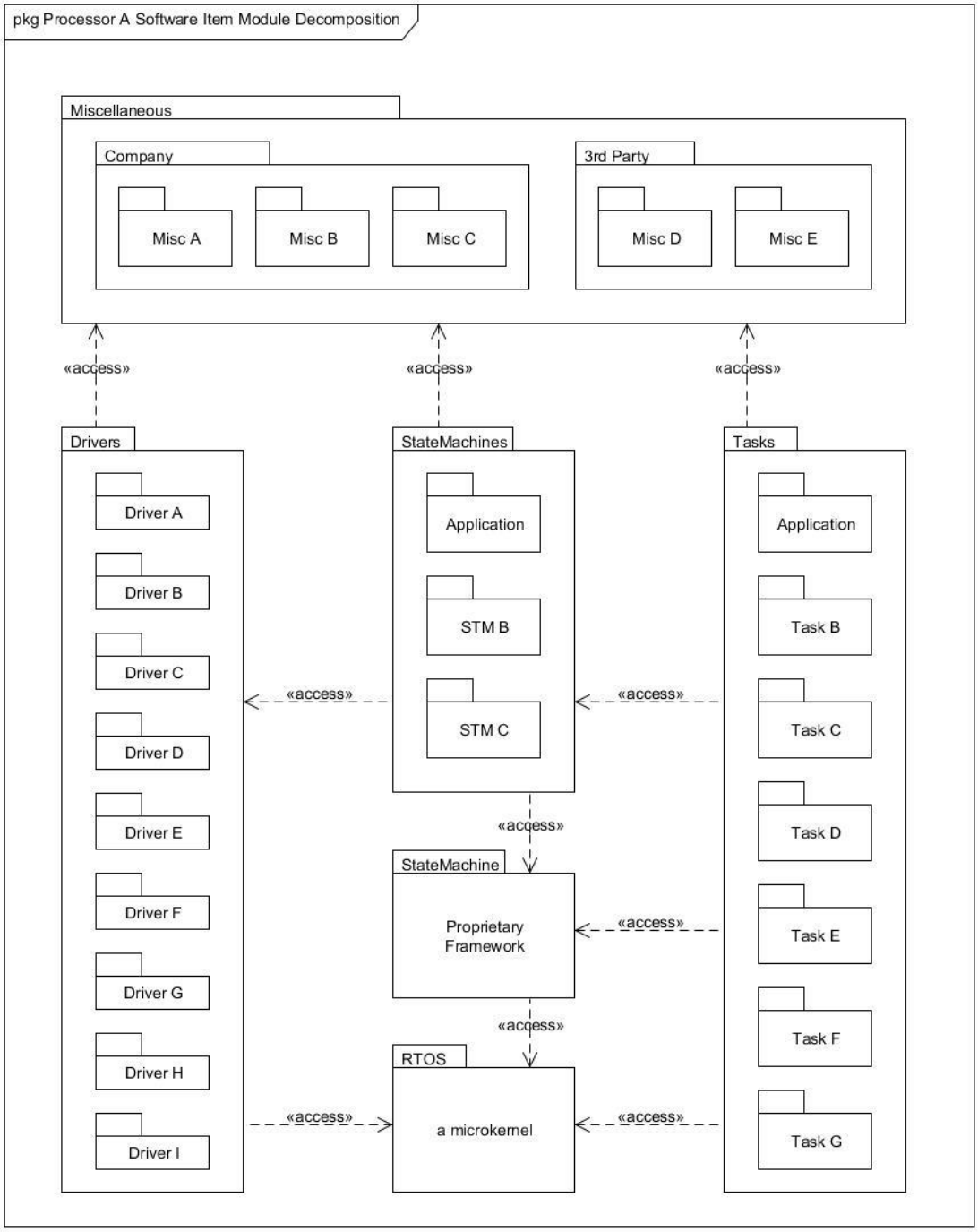


Figure A2.12. Module decomposition of software item of the “Processor A”. Notation: UML package diagram.

The foremost concept in the module decomposition of the *“Processor A”* software item is that tasks construct state machines, which are executed using proprietary state machine framework. The machines use drivers, which control hardware. The hardware include actuators, sensors, communication interfaces, memory interfaces etc. Interprocess communication between different active components is carried out using services of the underlying microkernel. Active components are tasks. Drivers that service

hardware interrupts are passive components. Package *"Miscellaneous"* contains supporting modules such as file systems, encryption/decrypting libraries and similar. The package contains also third party or SOUP components. Miscellaneous software items are accessed by the application software, meaning not by the kernel nor the state machine framework. The software item *"Processor A"* can be compiled with target environment compiler and with a simulation environment compiler. Environment specific differences are mostly handled in a pre-compiler level, but some software units, such as most of the drivers, have different versions for different environments.

### A2.3.3 Element Catalog

This element catalog describes the elements presented in the primary presentation of the view *"Development"*.

#### *A2.3.3.1 Tasks*

Task are concurrent processes scheduled by the microkernel. A task typically has a state machine software item as a component, in which all business logic is handled. The task software item only initializes the machine and dispatches messages to it from the interprocess communication bus of the microkernel. Chapter A2.4 explains the task architecture in detail. Some of the tasks may be machineless, such as background tasks intended to perform low priority calculations with the processor time not needed by the higher priority tasks.

#### *A2.3.3.2 State Machine Framework*

The company proprietary state machine framework supports most parts of the David Harels state machine formalism, which is the basis for UML state machine diagram convention. The supported features are hierarchically nested states, entry and exit actions, state actions, external and internal transitions, initial states, final states and shallow and deep history states. The main feature that is lacking is the orthogonal regions. Once a *"Task"* has initialized a machine, then all events to the machine are dispatched using the framework services. The machine may use callbacks from the task, but the task is not allowed to interfere the machine execution.

*A2.3.3.3 State Machines*

Essentially, the state machines are structures with current state and initial state variables and states are structures with parent state variable and event handlers. Using this data, the state machine framework is able to run the machines in the manner that is compatible with David Harels formalism. For the reason that the machines are structures, they always need an active component, a task, to run them. For instance, state machine *"Application"* is a component of the task *"Application"*. The state machine "*Application"* is the main state machine of the *"Processor A"* software item. Even though other machines are technically independent, they are in practice under control of the "*Application"*. The control logic of the *"Processor A"* software item is related to task architecture and is described in chapter A2.4. The UML models of the machines are described in the view *"Functional Description".*

*A2.3.3.4 Real Time Operating System*

The real time operating system in use is a microkernel with minimal set of features, mainly task scheduling, memory management and interprocess communication. The details of the kernel in use are presented in the software design description of the company project.

*A2.3.3.5 Drivers*

The drivers are software units that the software items use in order to control hardware. For example, each communication interface requires a driver. Especially many of the communication drivers have so much in common, that they are inherited from a base driver of a certain type of communication interface. The task architecture presented in chapter A2.4 describes how tasks accesses the drivers.

*A2.3.3.6 Miscellaneous*

The miscellaneous package is a collection of software items and units that the other software items require. Some of the items are developed by the company and some are originated from third party sources. The items and units include, for instance, a non-

volatile file system, a volatile file system, encryption and decryption algorithms and application specific mathematical algorithms.

### A2.3.4 Context Diagram

The context diagram of the view *"Development"* is presented in the Figure A2.13. The purpose of the diagram is to visualize the environment in which the *"Processor A"* software item is developed. The context diagram of *"Processor B"* software item development would be almost identical to the Figure A2.13. Only *"Simulator IDE"* would be excluded.

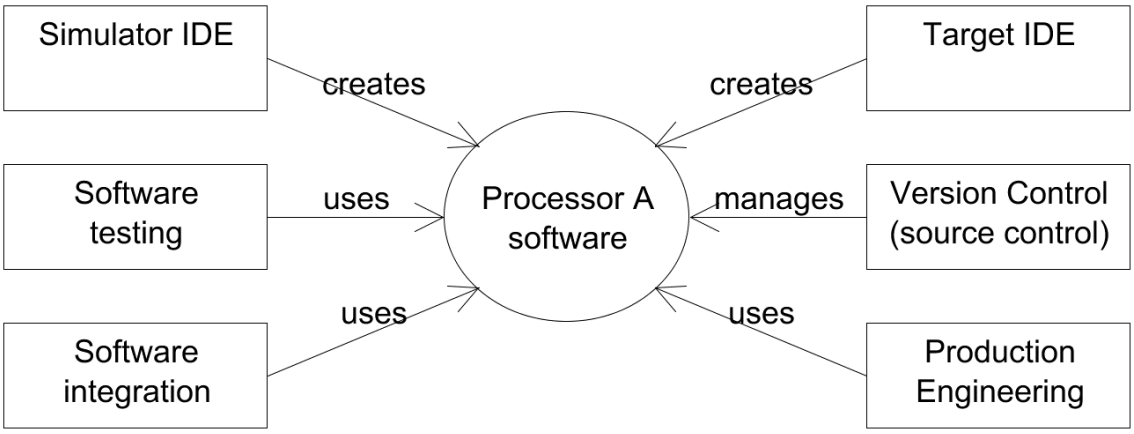


Figure A2.13. Context of "Processor A" software item development. Notation: Informal.

There are two separate integrated development environments (IDEs) involved in creating the *"Processor A"* software item. The *"Target IDE"* is for developing and compiling the binary that is executed in the real device. The IDE is also used for debugging the target software in the real environment. The *"Simulator IDE"* is also used for developing the software. It compiles an executable that is a simulated version of the actual target binary. The simulation enables faster debugging and unit testing compared to the real environment. However, not all functionality can be comprehensively simulated and especially troubleshooting of the software functionality in the hardware boundary has to be performed using the *"Target IDE"*.

Changes to the *"Processor A"* software item source code are managed in a version control system. The tool for change management is a one commonly used in the company. Software testing function of the company project uses both the target binary and the simulation executable, as many test cases related to the architecture can be

verified from the simulation. Software integration function uses simulated version in continuous building process for executing the units test every time a change is contributed to the change management system. The target binary is integrated to the whole software system and production engineering requires it for making manufactured un-programmed devices alive.

### A2.3.5 Variability Guide

Generic principle of software module organization is that a task executes a state machine. Exception to the rule is allowed if a task functionality is so trivial that the state machine model based system behavior description is not compromised. A state machine always required a task. A task can execute several machines. Software items must comply with coding guidelines of the company. Data hiding, encapsulation and layering are to be used where applicable. Programming model is event driven. Time consuming functions and execution blocking must be avoided.

### A2.3.6 Rationale

The development principles are selected to enforce rapid model driven development in a simulated environment. State machine models create visibility to the behavior and simulation enables faster development without discomfort and slowness of normal reality. Patterns used in the view are kernel pattern, layer pattern, event driven programming, model driven programming and simulation.

## A2.4 Task Architecture

The view *"Task Architecture"* describes how concurrent processes or tasks are organized in the software, how they communicate with each other's and what their execution priorities are.

### A2.4.1 Process Viewpoint

Figure A2.14 presents a *"Process"* viewpoint to the view *"Task Architecture"*. Stakeholder of the viewpoint is a software developer. The stakeholder is concerned how the task architecture organizes concurrent tasks and how reentrancy is designed; meaning in

which priority order tasks may interrupt other tasks. Another concern is how the tasks communicate with each other's.

| «stakeholder»<br>**Developer** |
| --- |
| concern="How is the concurrency of the software items designed?" |

| «viepoint»<br>**Process** |
| --- |
| stakeholder=Developer<br>purpose="Describe the concurrent task architecture and explain re-entrancy and communication scehems."<br>concern="How is the concurrency of the software items designed?" |

Figure A2.14. Viewpoint "Process" to the view "Task Architecture". Notation: SysML.

### A2.4.2 Primary Presentation

The primary presentation of the view is presented in the Figure A2.15 Notation for the presentation is UML communication diagram, which is a variation of UML interaction diagram.

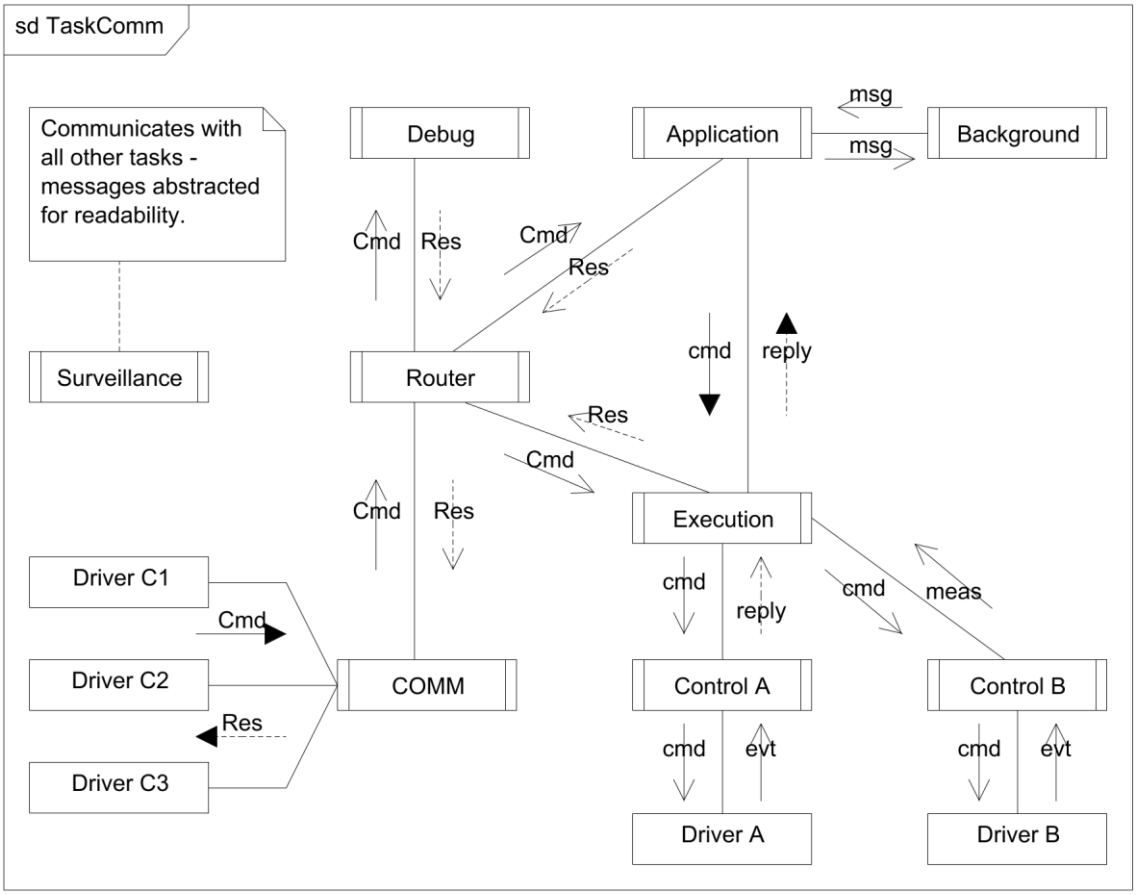


Figure A2.15. Communication scheme between tasks of "Processor A" software item. Notation: UML communication diagram.

The *"Processor A"* software item is controlled from external systems using synchronous command-response messaging. Drivers send spontaneous event messages, generated in hardware interrupt handlers, to the tasks. The diagram is for *"Processor A"*, the task architecture of the less complex *"Processor B"* software item is described in the software design description of the company project.

### A2.4.3 Element Catalog

This element catalog describes the elements presented in the primary presentation of the view *"Task Architecture"*. Each tasks listed in the catalog have a scheduling priority in the underlying pre-emptive real-time operating system. A task with a higher priority can interrupt execution of a task with lower priority. The execution is continued after the higher priority task has finished.

#### *A2.4.3.1 Task COMM and Drivers C1, C2 and C3*

Task *"COMM"* is a low priority task that reads and writes communication interfaces. Each interface has a *"driver"*, a software item capable of reading incoming data from a specific hardware. External systems connected to the interfaces send *"Cmd"* messages in a synchronous manner, expecting them to be answered with *"Res"* responses within a reasonable short time. An incoming command packet is passed to the task *"Router"* for further processing. No new data is given for the *"Router"* until it has acknowledged the previous one. A possible new data awaits in the data buffers of communication drivers. The responses are generated asynchronously in other tasks, the *"COMM"* task is not involved in message aggregation or handling. The responding tasks with higher priority must avoid suffocating the *"COMM"* task by lowering their priority when sending larger response streams. Assumedly, the task *"COMM"* does not run any state machines.

#### *A2.4.3.2 Task Router*

Task *"Router"* aggregates commands and forwards them to other tasks based on the types of the commands. Scheduling priority of the task is medium. Commands for development use only are forwarded to the *"Debug"* task. Commands that do not require changes in the application state are send to the task *"Execution"*. An example of such is a software item version query. All other commands are queued to the task *"Application"*,

which changes its state according to the commands. The *"Router"* task acknowledges the *"COMM"* task after every aggregated command if the command queue of the *"Application"* task is not full. Responses to external commands are sent using methods provided by the *"Router"* task. Task *"Router"* does not run a state machine.

### *A2.4.3.3 Task Application*

Task *"Application"* is the main controlling object in the hierarchical control architecture of the *"Processor A"* software item. The control is distributed to several other independent tasks also, but they still require stimuli from the task *"Application"*. The core of the application functionality is execution of sequences uploaded to the *"Processor A"* by external systems. A sequence is a payload of a *"Cmd"-command* and it translates to a stream of *"cmd"*-subcommands. Task *"Application"* sends these subcommands to task *"Execution"* in a synchronous fashion - next subcommand is sent after previous is replied. The task uses also task *"Background"* for time demanding low priority operations. Priority of the task *"Application"* is medium. The behavior of the task is implemented into a state machine structure from an *"Application"* state machine model.

### *A2.4.3.4 Task Execution*

Task *"Execution"* executes *"cmd"*-subcommands sent by tasks "Router" and *"Application"*. The ones received from the task *"Router",* by-passing the task "Application", are related to maintenance and similar purposes and do not affect to the main application functionality. Which is the execution of subcommand sequences. Nevertheless, all subcommands are either responded immediately or processed and forwarded to distributed controllers that control the hardware of the device. If a subcommand is not replied immediately, the reply to a commanding task is sent after a distributed controller has finished its part of the job. Task *"Execution"* runs a state machine that implements the subcommand execution functionality. The priority of the task is higher than the priority of task *"Application"*.

*A2.4.3.5 Control Tasks and Drivers*

Distributed controllers, such as tasks *"Control A"* and *"Control B"* execute state machines that control the actual hardware of the device, including actuators, sensors and similar. These tasks have high priority in the software item. All the jobs to be done are received from the task *"Execution"*. The controlling state machines use specific drivers to access the hardware. The drivers, such as *"Driver A"* and *"Driver B"*, have methods that interact directly with processor hardware registers and cause the surrounding electronics and ultimately mechanics to function as desired. The drivers are not tasks nor state machines, but they may have state-like existence in the processor hardware interrupt handler sub-routines. These routines are used to react to the chances in the external physical environment. The feedback to the distributed controllers is given using event messaging services of the underlying real-time operating system. The controllers interpret these events, for instance, as "job done" replies that are to be forwarded to the task *"Execution"*. Other type of an event is a value of a continuous measurement, which is passed spontaneously upwards in the control hierarchy without any *"cmd"-"reply"*-messaging, expect the one that sets the measurement process on/off.

*A2.4.3.6 Task Background*

Task *"Background"* supports task *"Application"* by performing calculations and file operations with low priority. The messaging related to the operations is handled using specific messages on the real-time operating system message bus. The task *"Background"* does not necessarily run a state machine as the nature of the operations is typically procedural.

*A2.4.3.7 Task Surveillance*

Task *"Surveillance"* monitors the sanity of other tasks and performs a system reset if any part of the software item becomes unresponsive or irrational. The task has the highest priority in the software item, meaning that it can interrupt execution of any other task. The task utilizes also a watchdog feature provided by the processor hardware. The watchdog is a hardware counter that causes a system reset if it succeeds counting to a specific value with a certain increment frequency. The sanity check for the task

*"Surveillance"* means that it should be able to "kick" the watchdog periodically in order to prevent the reset.

#### *A2.4.3.8 Task Debug*

Task *"Debug"* executes commands that are used for development, monitoring and testing purposes. The task run no state machine and it is executed with medium priority.

### A2.4.4 Context Diagram

NA. See primary presentation.

### A2.4.5 Variability Guide

Exact priority order of the tasks is not fixed. Especially the priority of the control tasks depend on real-time constraints that can be answered only during development time. The amount of control task can be extended whenever there is a need for new hardware features. New tasks should follow the same principles that are used for designing the initial control tasks.

### A2.4.6 Rationale

The view *"Task Architecture"* uses several architectural patterns. Application behavior, command execution and hardware control are layered to an increasing order of required real-time performance. The layering supports also hardware abstraction, as near hardware software is located in the drivers. The simulation version of the software item can use the same application code base as the target version – only drivers need to be implemented differently. The control logic is based on the hierarchical control pattern. The main active object in control is the task *"Application"*. Other tasks have different levels of independency, but require always some trigger from the *"Application"* in order them to change their initial state. Messaging between the tasks using services of the operating system is fundamentally asynchronous, but logically there is also some synchronization between the tasks. For the case of a deadlock in the command execution synchronization, the application has to provide a special command for external systems for resetting the states of the machines in all tasks. Many of the design decisions

in the task architecture are proven to be effective in previous projects of the company and similarity of the designs allows the project to utilize existing tested codebase.

## A2.5 Deployment View

The view *"Deployment"* describes the physical environment to which the executable software items are deployed. The environment in this description is limited to the device without the separate user interface module.

### A2.5.1 Physical Viewpoint

Figure A2.16 presents a *"Physical"* viewpoint to the view *"Deployment"*. Stakeholders of the viewpoint are concerned how and to where the software items are to be deployed. A typical stakeholder for the viewpoint is an integrator, a person whose task is to build a complete software system deployment package and methods for distributing the separate software items from the package to correct locations in the device.

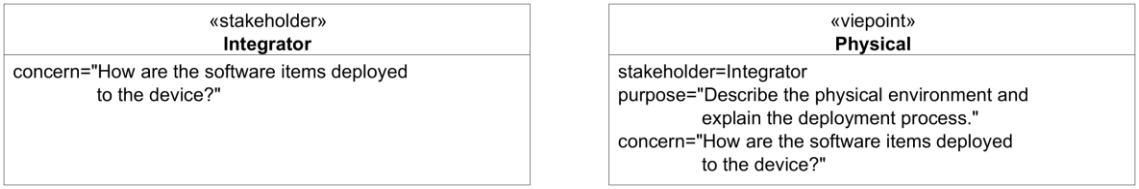


Figure A2.16. Viewpoint "Physical" to the view "Deployment". Notation: SysML.

### A2.5.2 Primary Presentation

The primary presentation of the view is presented in the Figure A2.17. Notation for the presentation is UML deployment diagram. The *"devices"* in the diagram are computational nodes that can execute and/or store artifacts. The devices are connected via communication interfaces to each other's and to external systems. The *"components"* are information containers such as file system folders or other data structures. The artifacts that are deployed across the devices and components are marked with a symbol that resembles paper sheet with folded upper right corner. Some of the artifacts are deployed during initial programming or software upgrade process. Other artifacts are transferred to the device or generated by the device during normal operation.

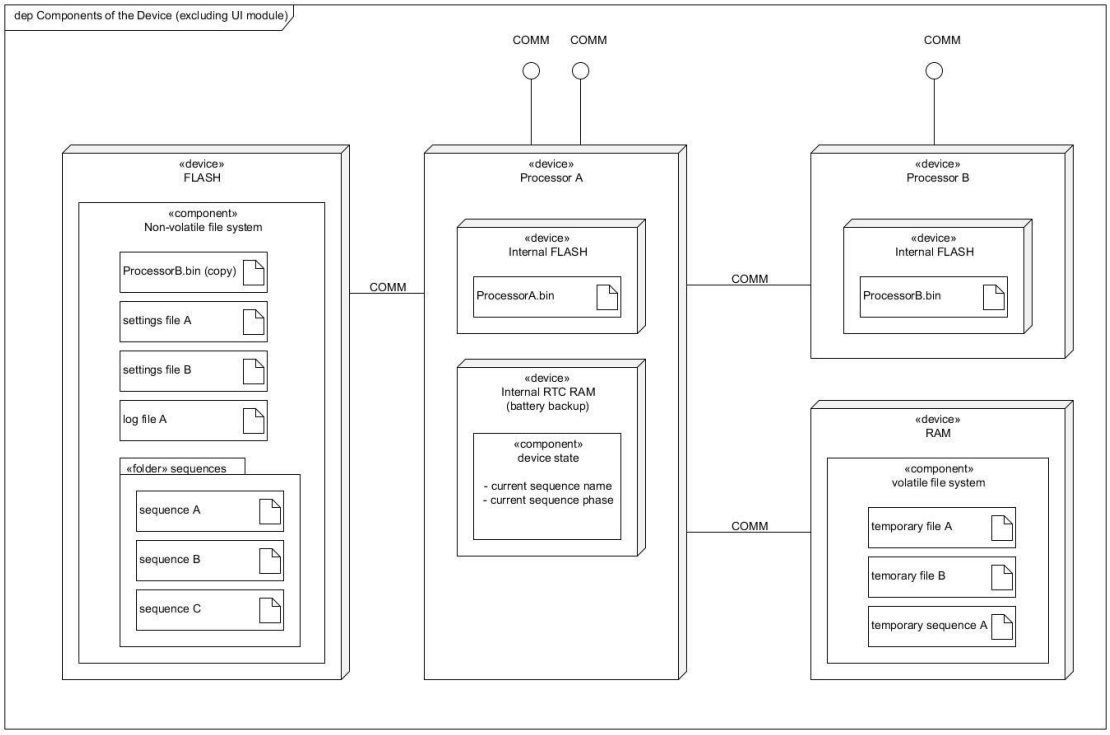


Figure A2.17. Deployment of processor A and B software items. Notation: UML deployment diagram.

### A2.5.3 Element Catalog

This element catalog describes the elements presented in the primary presentation of the view *"Deployment"*.

#### *A2.5.3.1 Processor A*

The device *"Processor A"* is a computational node that is capable of executing "*Processor A"* software item. The compiled item, artifact *"ProcessorA.bin"* is initially deployed to the internal memory of the device using specific interface of the device in a special mode of the processor. The special mode is enabled by altering voltage levels of certain inputs of the device. The initial deployment is performed during manufacturing. During initial deployment, a copy of the artifact *"ProcessorB.bin"* is transferred through the device *"Processor A"* to device *"FLASH"*. The artifact is a compiled binary image of the device *"Processor B"*. If the artifact is different from the one in *"Processor B"*, then the *"Processor A"* deploys it to the internal memory of *"Processor B"* using the communication interface between the devices. The artifact *"ProcessorA.bin"* can be upgraded after initial deployment by an external system connected to a communication

interface of artifact *"Processor B"*. The system is, for instance, a separate user interface module. The deployment in this case is done by the *"Processor B"* device, while the external system supports the process by providing the artifact as a data stream. The *"Processor A"* device stores also a current sequence backup artifact in its internal battery backed memory. This artifact is used in a sequence recovery workflow after a power failure.

*A2.5.3.2 Processor B*

The device *"Processor B"* is a computational node that is capable of executing "*Processor B"* software item. The deployment of the compiled item, artifact *"ProcessorB.bin"* is performed by the device "*Processor A",* if the artifact differs from the one that is stored in the device *"FLASH"*. The copy of the artifact in device *"FLASH"* is the latest version of the *"Processor B"* software item. It is transferred to the *"FLASH"* either during initial deployment or as a final step of the upgrade process of *"Processor A"* software item.

*A2.5.3.3 FLASH Device*

The *"FLASH"* device is a non-volatile memory device holding a file system for *"Processor A"* software item. The file system is required for storing various settings and logs. In addition, it contains a copy of the artifact *"ProcessorB.bin"*, which is used whenever the *"Processor B"* software item is deployed to device *"Processor B"*. There is also a folder *"sequences"* in the file system. The folder is for permanently storing application sequences uploaded by external systems via the communication interfaces of the device *"Processor A"*.

*A2.5.3.4 RAM Device*

The "RAM" device is a volatile memory device with a file system for temporary files. The file system is used by the *"Processor A"* software item for temporarily storing sequences to be executed and for other miscellaneous temporary files. The sequences are uploaded by external systems via the communication interfaces of device *"Processor A"*. After the

execution, the sequences are lost. When the device is powered off, then also all other files in the temporary file system are lost.

### A2.5.4 Context Diagram

The context diagram of the view *"Development"* is presented in the Figure A2.18. Deployment of the executable binaries is performed initially during manufacturing and later during software upgrade process. The software upgrade is initiated by the internal UI module of the device. The sequences, temporary or permanent, are deployed to the device during normal operation by different systems connected to the communication interfaces of the device.

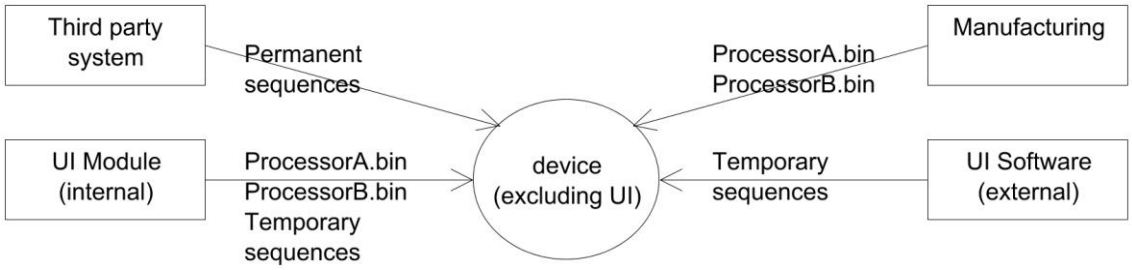


Figure A2.18. Context of the device software deployment. Notation: Informal.

The context diagram in the Figure A2.18 excludes development activities. Software developers have means to deploy binaries to the device using interfaces that are not available for other users. Development function is also responsible for utilizing the *"prevent reverse-engineering"* tactic, meaning that reading back the binaries from the device should be made as difficult as possible. The developers are also capable of altering the contents of the file systems.

### A2.5.5 Variability Guide

The executable binaries can be deployed either using the manufacturing process or utilizing the software upgrade functionality provided by the separate UI module. The amount of permanent application sequences is limited by the available memory size in the non-volatile file system. Only one temporary sequence can exists at a time in the device.

A2.5.6 Rationale

The deployment design is done within the limits defined by the system architecture. The purpose of the design is to enable a capability to initialize and upgrade the computational nodes of the device. In addition, the view explains how the different artifacts required or generated during normal operation of the device are distributed around the components of the device. This supports development, maintenance and testing of the system. Holding a copy of the software item *"Processor B"* is part of the strategy to elevate availability of the device by ensuring that deployment is valid and does never fail. In addition, the view addresses quality attribute *"security"* by applying tactic *"prevent reverse-engineering"*.

## A2.6 Scenarios

Scenarios is the fifth view of the Kruchten's 4+1 architecture model and the sixth view of the architecture at hand, as the model was extended with the view *"Executive View"* in this description. The description of the scenarios does not use the view template. Instead, it is a package of UML use case and interaction diagrams. This description includes some of those diagrams. The complete package is described in the software design description of the company project. Figure A2.19 presents three main use cases of an external UI software, which connects to one of the communication interfaces of the device.

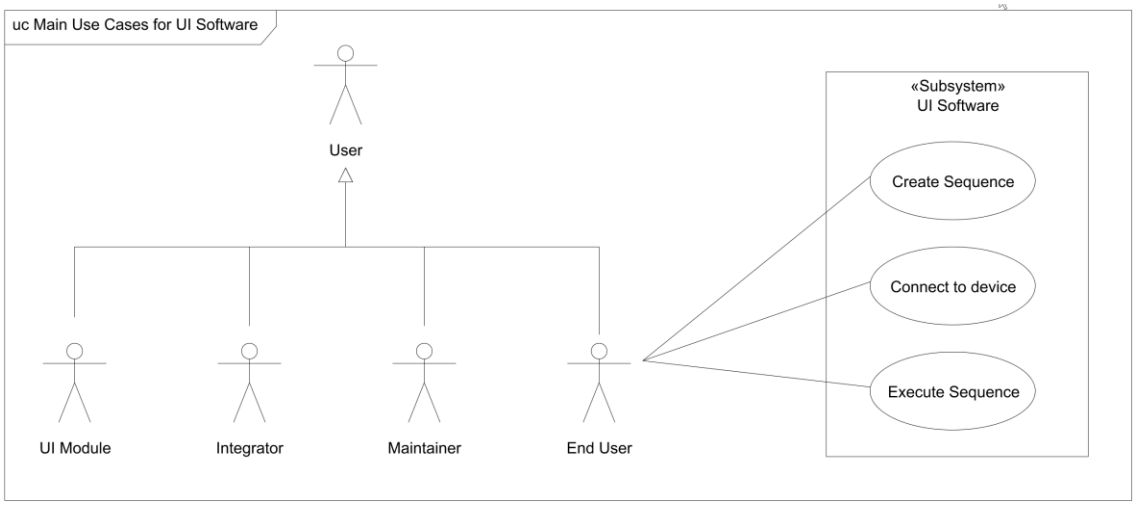


Figure A2.19. UI software use cases. Notation: UML use case diagram.

The external UI software is used for editing sequences and executing them in a device. The user of the software is an *"End User".* In Figure A2.19, the *"End User"* is inherited from a *"User"*, which is a base for all user types. A user for the device can also be another system, a non-human. For the device, the internal separate UI module is a user. The uses cases for the *"End User"* are the ovals inside the *"UI Software"* subsystem boundary. Figure A2.20 decomposes the use case *"Connect to a device"* into a series of interactions between different software items in the system. The notation in the figure is UML sequence diagram, which is a variant of UML interaction diagram.

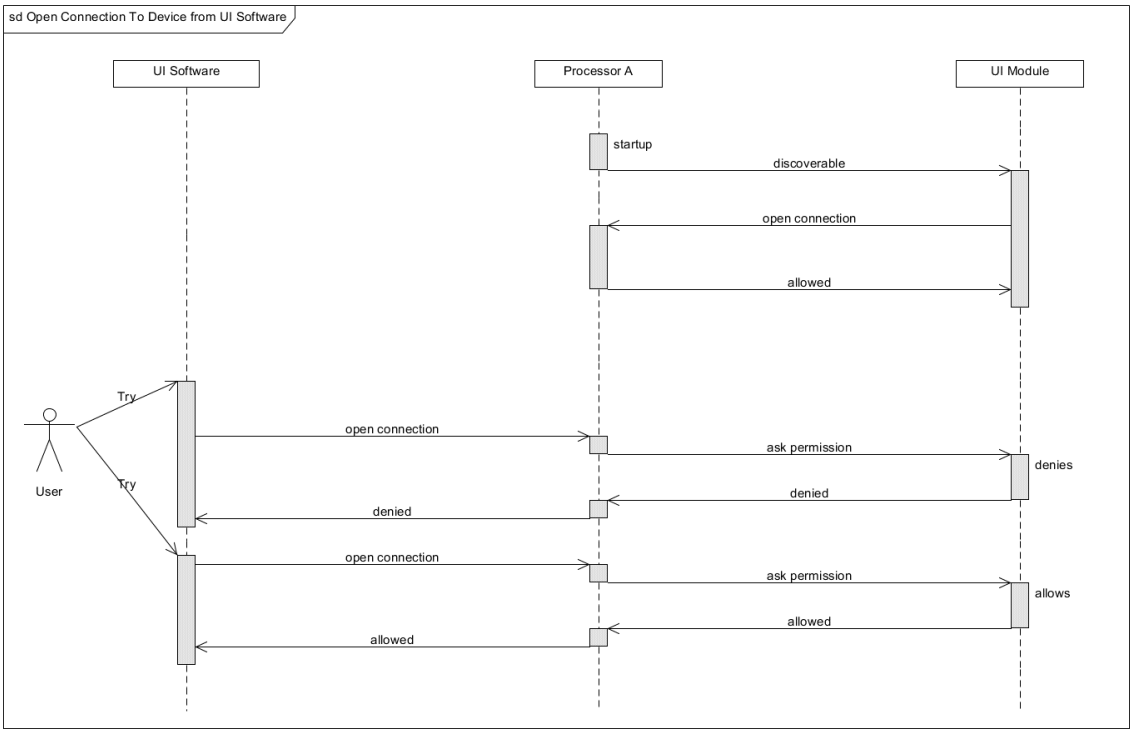


Figure A2.20. Interactions of "Open Connection" use case. Notation: UML sequence diagram.

Sequence diagram in the Figure A2.20 presents lifelines of "UI Software", "Processor A" and "UI Module" in a use case where a user tires to open a connection from the "UI Software" to the device. The device is a combination of the "Processor A" and the "UI Module". Before the user action, the "Processor A" has already announced itself as discoverable for the "UI Module". This is done in the power-on startup routine. In addition, the "UI Module" has opened a connection to the "Processor A" after the discovery. In the "UI Software" lifeline, the first user connection attempt fails, because the "UI Module" denies the permission to open the connection. The second time the user succeeds, as the "UI Module" allows the connection. The scenario is one of the many that describe how the control of the device is reserved and released between different users. As usual,

the rest of the scenarios can be found from the software design description of the company project.

## Appendix 3. SOUP Identification Template

| SOUP Identification Template | |
|---|---|
| **Title** | |
| **Manufacturer** | |
| **Designator**[1] | |
| **Description (optional)** | |
| | |

[1] IEC 62304 chapter 8.1.2: *"The unique SOUP designator could be, for example, a VERSION, a release date, a patch number or an upgrade designation."*

# Appendix 4. Interviews – Data

| Interview Data | |
|---|---|
| **Interview #** | **1** |
| **Title of the interviewee** | **Senior Project Manager** |
| **Introductive Questions** | |
| 1.1 | What is your overall impression of the study?<br>*"The thesis comprehensible addresses the basics of software architecture in regulated environment. Interfaces between different software items were documented nicely. Background chapter is heavy reading, because it discusses regulations et cetera. However, the key issues are well covered. Conclusions were not finished yet. I'm eagerly waiting for those."* |
| 1.2 | What is your relation to the project/thesis?<br>*"I'm a project manager of the company project related to this thesis."* |
| 1.3 | What expectations did you have towards the study and/or the architecture?<br>*"I was interested to know more about the specialties of IVD MD product development."* |
| **Detailed Questions** | |
| 2.1 | What is the level of significance (low, medium or high) of the study? Explain.<br>*"High. Justification for the given level is that we have to know and be able to describe what we are doing, in the light of the standards. And to be able to justify our decisions. That is why this is important in our operating domain. Justifications have to exist and they need to be derived from the standards. We need to be able to rely on them and be able to return to them later. This is why I find the work to be significant."* |
| 2.2 | Was the goal of the study clear? Was it achieved?<br>*"It was. The thesis describes the requirements of the regulated environment well, and from different perspectives. That topic is covered well enough. On the other hand, the thesis has been able to delimit very wide topic to reasonable. The goal was to clarify requirements of the standards, analyse our current operation procedures and investigate how we should improve our procedures. Seems that the goal was achieved, but I'm still waiting for the conclusions for the final recommendations."* |
| 2.3 | Is the background information comprehensive enough?<br>*"The topic is wide and one should always find the key elements. I think that the thesis succeeded to discuss the essential topics. I think it was good that the thesis did not replicate the standards but interpreted the essential things."* |
| 2.4 | Is the structure of the study clear?<br>*"The thesis was well structured. The figures made the reading of the heavy text to much easier. Generally, the thesis managed to present the subject in an understandable form."* |
| 2.5 | What is the usefulness of the study for you or your organization?<br>*"The work is useful for us as it is. The company confidential information is nicely hidden in the thesis, so the work is useful also to public audience. In addition, we have already been able to utilize the knowledge that has been evolved from the thesis work."* |
| 2.6 | What kind of opportunities does the study create for future development?<br>*"The results of the thesis will form a basis for the future work, especially when we renew our operating procedures for software development. It will be the starting point for the discussions for the future development. "* |

| Other comments: |
|---|
| *"All in all the work is good. The structure of the thesis is good and the text is easy to read. If the text could be compressed even more, then it would be even easier to read. However, with the topic so wide, if you omit too much, then the work would only scratch the surface of the domain. Actually, I would not necessarily recommend compressing the text. "* |

| **Interview Data** | |
|---|---|
| **Interview #** | **2** |
| **Title of the interviewee** | **Senior R&D manager** |
| **Introductive Questions** | |
| 1.1 | What is your overall impression of the study?<br>*"A comprehensive summary of the standards related to software development and to other areas also. IVD vs. non-IVD topic is highlighted. Somehow confusing, because there is some much text and the subject is heavy. As a visual person, I would have liked to see even more figures to clarify the context to which the different topics are related. Huge amount of standards and directives. How do they really relate to each other's and how can I be sure that we have taken into account all standards? Perhaps it is evident in the text. I was focusing to the big picture when reading the thesis. Overall impression of the thesis is very good; well written, good English, well structured."* |
| 1.2 | What is your relation to the project/thesis?<br>*"As a software team manager, I'm responsible for software processes and standard operation procedures of software development. The fact that the SOPs are followed is also my responsibility. In addition, I am managing the software verification activities in our organization."* |
| 1.3 | What expectations did you have towards the study and/or the architecture?<br>*"I expected clarification to the issue of what standards should we really follow in our software development process, which includes software testing. Having this answered we are able to start discussions on how to update our SOPs."* |
| **Detailed Questions** | |
| 2.1 | What is the level of significance (low, medium or high) of the study? Explain.<br>*"High. Definitely. Since the thesis analyses our software development through a real product development project, we are able to find out if something essential is missing from our processes. This is important, because we need to be sure that when we say that we follow regulations and standards, that then it really is true."* |
| 2.2 | Was the goal of the study clear? Was it achieved?<br>*"In the beginning (of the study), I wasn't. However, now that I have read the latest version, I think I see it. The goal was to investigate, from the viewpoint of the software architecture, which standards and regulations affect the design. And how the design should take the standards and regulations into consideration in the way that it complies against the standards and regulations during the whole development phase. And the basis for this is the architectural design. I think the goal was achieved. In addition, I think the architect himself has understood that every architect should have enough knowledge on the subject in order to be capable of considering the subject in the beginning of the design phase. It is difficult to fix afterwards."* |
| 2.3 | Is the background information comprehensive enough?<br>*"I need to revisit the background chapter for clarifying the thing that do we know which standards we should really follow (see 1.1)."* |

| | |
|---|---|
| 2.4 | Is the structure of the study clear?<br>*"I'm not quite sure if I can see the goal of the thesis from the structure… (interviewee is browsing table of contents) but it is ok. The background chapter is massive, yet it is clear and the most interesting one. Then comes the materials and methods and results… yes the structure is good in its whole. It is just that the subject is so large. Still, all parts are necessary from my opinion."* |
| 2.5 | What is the usefulness of the study for you or your organization?<br>*"On the behalf of myself, as we are planning to clarify our current SOPs, the thesis is very useful and comes at good time, because now we can refer to it when updating the SOPs. What it means to others… perhaps they benefit from the updated SOPs or directly from the thesis."* |
| 2.6 | What kind of opportunities does the study create for future development?<br>*"Same thing what I answered to the previous question. On the other hand, the thesis focuses on EU medical regulations and standards. For future development, we could investigate the regulations of other market areas more thoroughly. In addition we could, clarify or update the requirements of ISO 9001 quality system for software development."* |
| Other comments:<br>*"On the whole, very good thesis. The company benefits."* | |

| **Interview Data** | |
|---|---|
| **Interview #** | **3** |
| **Title of the interviewee** | **Workstation Software Architect** |
| **Introductive Questions** | |
| 1.1 | What is your overall impression of the study?<br>*"The work is pretty ok and clear. First regulated environment, then, logically, what kind of architectural methods are available for complying the regulations. Finally, in more detail, how things are taken into consideration in our ongoing project. Logical sections, that is. On the other hand, the numerous amount of different abbreviations made it a bit difficult to read, some of them could have been re-opened in subsequent chapters."* |
| 1.2 | What is your relation to the project/thesis?<br>*"I'm part of the project team, participating in design and development of higher level software items (UI software). In addition, I participate in system design."* |
| 1.3 | What expectations did you have towards the study and/or the architecture?<br>*"Originally, I thought that this thesis was about how the regulations are taken into consideration generally in our field. And how those have changed recently. Only afterwards I realized that the thesis discusses our project also."* |
| **Detailed Questions** | |
| 2.1 | What is the level of significance (low, medium or high) of the study? Explain.<br>*"Depending on the target audience… but for me it was interesting for the reason that I seldom have to think these regulations and such.* I think these subjects concretised to me in a generic level. I think this work is valuable for any software architect or developer who needs to study the subject. The significance is high.*"* |
| 2.2 | Was the goal of the study clear? Was it achieved?<br>*"Perhaps the goal was explained in the beginning... Was it about describing our project, describing architectural methods or the background or all of them together? I assume it was the description of our project. (objectives are read to the interviewee from the introduction) Ok, then yes, all the three goals were actually achieved."* |

| | |
|---|---|
| 2.3 | Is the background information comprehensive enough?<br>*"I think there is enough material in the background, truly so. The regulated environment is difficult to digest because of all the various aspects of it. Perhaps a wider handling of the subject wouldn't have improved that."* |
| 2.4 | Is the structure of the study clear?<br>*"Yes it was. Background, theory and practice"* |
| 2.5 | What is the usefulness of the study for you or your organization?<br>*"I could image that I could use the work when looking for answers to the topics that the thesis discusses. For the company… I think when the educated (by the thesis) employees create designs, then the company benefits from the increased quality."* |
| 2.6 | What kind of opportunities does the study create for future development?<br>*"I think in future there is less confusion around the subject. We have now an opportunity for clarifying our guidance for projects."* |
| Other comments:<br>*(interviewee gives feedback on miscellaneous grammar and spelling errors)* | |