Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

New Evaluation Method for Reference Architectures Utilized to Design a Software Reference Architecture for Liquid Handling Devices

Mohammad El Naofal

Course of Study:	Computer Science
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Andreas Bitter, Valentin Busch
Commenced:	April 21, 2021
Completed:	October 21, 2021

Abstract

Software development is important to ensure the system's requirements and qualities. If the architecture is implemented poorly, then the system might not be fulfill the users' expectations. A reference architecture can facilitate the process of achieving a concrete architecture because it provides architectural approaches, designs, and components as starting point that can be followed. A reference architecture can be helpful in the laboratory device domain especially for liquid handling devices because of the common requirements and qualities that the devices share. To reach a reference architecture, it needs to be evaluated. Unfortunately, there is not an evaluation method that is oriented towards evaluating reference architectures. As a result, this thesis presents an adapted evaluation method from Architecture Tradeoff Analysis Method (ATAM) to evaluate reference architectures. The adapted evaluation method was tested by applying it to the liquid handling device domain to obtain a reference architecture. The obtained reference architecture is also presented in this thesis and is tested by using it as a facilitator to implement a small prototype of a liquid handling device.

Contents

List of Figures

List of Tables

Acronyms

ALMA Architecture-Level Modifiability Analysis. 25

ALPSM Architecture Level Prediction of Software Maintenance. 57

ARID Active Reviews for Intermediate Design. 58

ASAAM Aspectual Software Architecture Analysis Method. 57

ATAM Architecture Tradeoff Analysis Method. 2

Architecture Tradeoff Analysis Method for Reference architectures. 57

CBAM Cost Benefit Analysis Method. 57

CIPA Creative Innovative Patterns for Architecture Analysis. 57

DoSAM Domain-Specific Software Architecture Comparison Model. 26

EBAE Empirically-Based Architecture Evaluation. 57

ESAAMI Extending SAAM by Integration in the Domain. 57

FAAM Family Architecture Assessment Method. 22

ISAAMCR Integrating SAAM in Domain-Centric and Reuse-Based. 58

SAAM Software Architecture Analysis Method. 18

SAAMCS SAAM for Complex Scenarios. 56

SAAMER Software Architecture Analysis for Evolution and Reusability. 24

SACAM Software Architecture Comparison Analysis Method. 57

SALUTA Scenario based Architecture Level Usability Analysis. 22

SBAR Scenario-Based Architecture Re-engineering. 57

SPE Software Performance Engineering. 57

SRA Software Reference Architecture. 8

UML Unified Modeling Language. 34

1 Introduction

Software architecture is a crucial part of any system. It helps technical and non-technical people understand how the system will be implemented and deployed. If the software architecture is not designed properly, poor architectural decisions in fulfilling requirements and qualities can be seen. Furthermore, software architecture can be time consuming if it has to be designed from scratch for each system, even if the systems share a similar domain. Thus, having a Software Reference Architecture (SRA) for a particular domain can provide benefits to reduce the time and effort needed to achieve concrete systems within a domain. It can act as a facilitator to ease the process of achieving concrete software architectures since the common requirements and qualities will be already handled by the SRA.

To achieve a good SRA an evaluation method needs to be used. There are many architecture evaluation methods, the most well known one is ATAM. However, none of them are specific towards SRAs which can lead to some disadvantages. As a result, the paper presents an architectural evaluation method that was adapted from ATAM. The adapted evaluation methods guide the architects and the software team to reach an SRA that can fulfill the requirements and qualities that the stakeholders expect from a domain.

Furthermore, to prove the effectiveness of the adapted evaluation method, it was applied on the laboratory device domain in particular for liquid handling devices. It resulted in an SRA for liquid handling devices which is also presented in this paper. I.DOT One is a laboratory device which was used as a test subject to test the effectiveness of the SRA by trying to reach a concrete architecture and implementing a small prototype using the SRA. In the end, the paper mentions future work that can be done.

1.1 Problem

Currently a software architecture has to be developed from scratch for each laboratory device even though they share some of the basic usage scenarios and functionalities such as data sharing, remote access, and automation. The development of a software architecture for each system within a particular domain is time-consuming because common software features have to be evaluated more than once. Common features such as how the user interacts with the devices, automation, security, connectivity, cloud integration, and maintainability.

A software architecture needs to be designed to implement and fulfill the required functionalities, quality attributes, constraints, and principles [Bro18]. If one of these requirements wasn't considered, then multiple issues might arise during and after the development of the software application which can affect response time, security, and maintainability [Bro18]. Moreover, poorly designed software

architecture will cause the business logic, user interface, and hardware code to be strongly coupled. Unnecessary strong coupling makes it harder to maintain, enhance, and understand the software because of the dependencies they introduce.

Problems from poorly designed software architecture can be seen in any domain. They can for example affect the maintainability, which is the ability of the architecture to cope with unexpected scenarios after designing or implementing the system. For example, in the laboratory device domain, new common features need to be introduced to a developed device. The new features are for example remote accessibility, automation, or cloud integration. To enable these features, several technical decisions and risks have to be evaluated such as communication protocols, security issues, and hardware constraints. If not dealt with properly, inadequate architectural decisions are taken.

1.2 Goal

The main goal of this thesis is to develop an SRA for a specific subset of laboratory devices which are liquid handling devices. It shall act as a facilitator to reach the concrete software architecture. An SRA facilitator acts as a base for future systems [CMV+09]. It is a starting point for architects to prevent the reinvention and revalidation of common scenarios and quality attributes in a particular domain [CMV+09]. An SRA should cover all the common scenarios and quality attributes of a laboratory device subset. It shall be easily adapted, if needed, to fit a specific device and to construct the concrete software architecture.

However, before introducing an SRA for the laboratory device subset, different architectures need to be compared against each other. This will be performed using an evaluation method to assess the architectures and obtain the most suitable one. The evaluation method indicates whether an SRA is applicable for a domain and if it covers common scenarios and quality attributes. In addition, depending on the SRA evaluation, multiple SRAs might be introduced to serve different laboratory device domains. Based upon the need of architectural comparison, the evaluation method for SRA will be introduced first in the thesis.

To achieve the goal, two research questions need to be answered. These are:

- **RQ1**: Is there an existing evaluation method that can be used to evaluate SRA for laboratory devices? Can existing methods be adapted to fit this specific case?
- **RQ2**: What is a suitable SRA for liquid handling devices, measured by the previously determined evaluation method?

1.3 Contribution and Research Methodology

The contribution behind this thesis comes from several factors. The first one is to solve the time-consuming software design and development of laboratory devices that is faced with companies and to provide a base on how to implement automation, cloud connectivity, and remote access for laboratory devices. There is not a lot of papers that discuss laboratory device software architecture more specifically very few talks about liquid handling devices. The second contribution is to fill a

research gap that is found with architecture evaluation methods. Not a lot of papers discuss how to evaluate reference architectures or present an evaluation method that is oriented towards designing and analyzing a reference architecture.

Figure 1.1 shows the research methodology that is going to be followed in this thesis to answer the research questions.

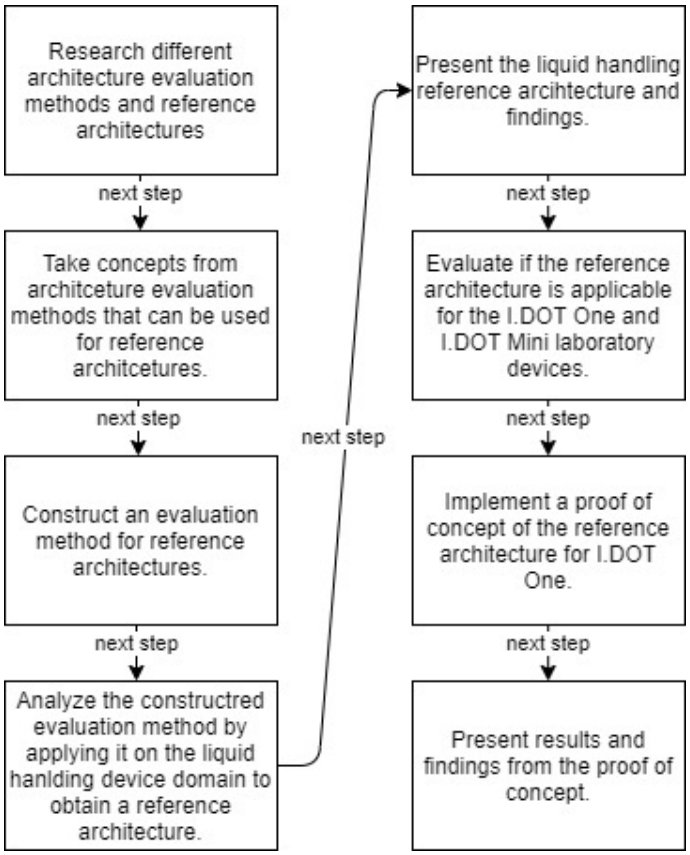

Figure 1.1: Research Methodology

2 State of the Art

There is a common problem for automating laboratories [KYZC12]. Systems are designed without considering integration with other systems or devices [KYZC12]. This can make it tedious trying to integrate different systems together. Some research have been made to standardize the communication and data between laboratory devices [PSL+20] [20]. One of the well-known ones is SilA2 [PSL+20] [20] [SA17]. Kong et al. [KYZC12] mentions that a lot of accessories can be added to a liquid handling device, which increases the importance of planning requirements to achieve a system with a friendly user interface that is easy to use and can integrate with other devices. Other research papers provide an architecture for an entire laboratory that has multiple devices. The architecture specifies how the devices can be automated and integrated with each other. For example, [KGFČ15] presents an architecture for automated control of remote laboratories. Another examples are [ZSZ11] and [Bis13] which present an architecture to make a laboratory virtually (remotely) available. There is also paper [SA17] that provides a centralized reference architecture to integrate different laboratory devices such as liquid handling devices. The reference architecture is supposed to address technical issues when adding platforms in an automated laboratory [SA17]. All other software components such as the user interface and data transfer will integrate with the main centralized system via plugins [SA17].

Unfortunately, there are not a lot of public documentation about problems faced with software development for liquid handling devices. The problems occur within the companies. Problems such as what frameworks, architectural patterns, and data formats should be followed. None of them specify the software architectural structure of liquid handling devices. They cover the entire laboratory without focusing on a single laboratory device or a certain kind of laboratory devices. They offer an architecture that requires the customer to host a centralize machine that will integrate with multiple devices. This adds complexity to the customer especially if they don't have a lot of laboratory devices and if they do not want certain aspects of the architecture such as remote communication or automation. Also, having one database for all devices can also increase the complexity of the system since different laboratory devices require different schemas. In respect to the laboratory device, the presented architectures are generic and does not provide different views of the architecture. As a result, this thesis plans to fill this gap and provide a detailed reference architecture for liquid handling devices.

3 Understanding Software Architecture and Software Design

Before introducing an SRA for laboratory devices, this chapter will briefly define the terms software architecture, software design, and reference architecture since there isn't a commonly agreed definition for them.

3.1 Software Architecture

A software architecture definition is going to be provided first followed by the software architectural drivers.

3.1.1 Definitions

Several definitions of software architecture have been proposed. The common requirement among them is to ensure that the overall vision and goal of the system is satisfied. Software architecture, if followed accordingly, determines the final system outcome. At first, four software architecture definitions are mentioned, followed by a chosen definition based on the knowledge from the four definitions and their correspondence to an SRA:

- The first definition is from the Software Engineering Institute [CBB+10] which is also mentioned by Fairbanks [Fai12]. It states that "software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."
- The second definition is from Solms [Sol12] where he defines software architecture as the "software infrastructure within which application components providing user functionality can be specified, deployed and executed ... Application components are software components which address functional requirements of the software system."
- The third definition is the definition from [RW12]. The authors define software architecture as a system that is made up of system elements and relations between them, system properties that are composed of system functionalities and quality attributes, and design principles that act as guidelines to the architecture's definition.
- The last and fourth definition is from [Bro18] where the author defines software architecture as a "combination of application architecture and system architecture, again in relation to structure and vision." Application architecture describes how the application is organized, designed, and built. System architecture has a higher view where it is composed of many deployable unites put together to understand the software and hardware.

All the definitions are about reaching the system's goal and delivering the stakeholders' vision. The first and third definition are the most suitable ones for this paper since they consider all the different architectural aspects that should be modeled in an SRA. The architectural aspects are: software elements, relations between them, and properties of each to describe a system's structure. These architectural aspects describe how the system is organized, designed, and built to deliver the required system's functionalities, quality attributes, scenarios, constraints, and principles that are provided by the stakeholders. The deliverables of the software architecture (functionalities, quality attributes, scenarios, constraints, and principles) are also the architectural drivers that shape the final software architecture.

3.1.2 Architectural Drivers

Architectural drivers are composed of five elements that the system needs to deliver. They are briefly explained below.

Functional Requirements

Use cases and functional requirements are often the first aspect that is discussed since they provide an overall understanding of the system's main goals. They specify the system capabilities and the end-users. System capabilities provide stories of what the end-user can achieve with the system [Bro18] [RW12].

Quality Attributes

Quality attributes, also known as non-functional requirements, specify what qualities the system should deliver. Quality attributes are for example performance, security, maintainability, usability, stability, understandability, and changeability. System requirements can be implemented using different architectural decisions. However, each architectural decision satisfies the quality attributes differently because of the different tradeoffs and risks they introduce. Consequently, it is important that these quality attributes are discussed, captured, and prioritized by stakeholders to give reasons behind following a decision [Bro18] [RW12] [Fai12].

Scenarios

A scenario is a short description of a stakeholder's interaction with the system [KKC00]. A designed software architecture should be evaluated on how well they satisfy the scenarios. According to Kazman et al. [KKC00], scenarios can be divided into three different types: Use case, growth, and exploratory scenarios. A use case scenario is similar to the functional requirements defined previously. They specify the different use cases that the end-user can have with the system. A growth scenario is a scenario that anticipates future changes that may occur to the system. An exploratory scenario is a scenario that explores extreme changes to the system. This is done to stress the system and test its capabilities under unusual circumstances [RW12].

Constraints

Constraints specify system architecture limits. They are often given by stakeholders, and they set boundaries on how well functional and non-functional requirements can be captured. Constraints can be time, cost, technologies to be used, and the organization or team implementing and developing the system. Constraints can be based on the team's familiarity with certain technology domains limiting the options that can be utilized to implement the architecture [RW12] [Bro18].

Principles

Architectural principles are rules and fundamentals that are followed to guide the design and implementation of the system. Principles can act as a good starting point to transition from the problem space to the solution space while providing reasons behind the architectural decisions. [RW12]. Architecture principles could be the type of architectural style that is going to be followed. For example, an N-tier layered architecture, pipes-and-filter, or a data center architecture. Moreover, architectural principles can also refer to the concepts that the architecture should have, such as coupling or the statefulness of the components. They can also set the communication protocols and formats that should be followed such as HTTP, XML, SOAP, and JSON. [Bro18] [RW12] [Fai12]

3.2 Distinction between Software Design and Software Architecture

Software architecture is software design but not vice versa [Bro18]. Software design is about choosing a solution from the many possible solutions of implementing the system [Bro18]. An architect should focus more on quality attributes than functionalities when designing a software architecture because functionalities can be achieved using different software design methods with different qualities [Fai12]. Software design dives more into lower-level scope of how the components are built.

To give a clear separation between software architecture and software design, software architecture should consist of the application significant decisions that are not easily changeable at later software life cycles [Bro18]. Software architecture is more important than software design in handling system related issues, like unexpected usage or change scenarios and quality attributes, because the system structure reflects on these issues [Fai12]. Software architecture embodies the details required to achieve an overall quality attribute of the system, on the other hand, software design is the implementation solution to deliver the functionalities and satisfy the quality attributes.

3.3 Software Reference Architecture

Similar to software architecture and design, there is no commonly agreed definition about SRA [CMV+09]. However, SRA is commonly known as a standard or a facilitator to achieve a concrete software architecture. It is designed at a higher level of abstraction compared to concrete software architecture [AGG12]. This makes an SRA applicable as a standard or a facilitator to multiple

different contexts within a similar domain. A standard SRA aims at system interoperability while the facilitator aims at providing inspirations and guidelines for the concrete software architecture design [AGG12].

Cloutier et al. [CMV+09] and Angelov et al. [AGG12] define SRA as a goal that captures existing architectures and domain knowledge to assist in development by implementing the shared functionalities and data flow of the software architecture. An SRA can also be seen as a high-level software architecture that captures common functionalities, quality attributes, constraints, and principles for a specific domain.

Benefits of SRA are:

- Reduce development costs and time [MAFA13] [MSA+15].
- Improve the architecture understandability and communication among teams and stakeholders [CMV+09] [MAFA13].
- Reduce risks [CMV+09].
- Increase overall software quality [MAFA13] [MSA+15].
- Eliminate revalidation and reinvention of solutions since it provides a standard to facility design and development [CMV+09] [AGG12].
- Can be well adapted in companies and organizations that are expanding with multiple similar software application projects [MSA+15].
- Suitable for architecting a given application domain to define the common architectural drivers including the quality attributes [MSA+15].

Drawbacks of SRA are:

- Introduces a learning curve for architects and developers if they are not familiar with SRA [MSA+15] [MAFA13].
- Initial time investment in building the SRA can be a problem if the system needs to be deployed in a short time period [MAFA13].
- Dependency issues on the SRA because it must be changed first before applying it to the systems if a new common architectural driver is introduced [MAFA13].
- Limited flexibility on systems because it is constrained to a specific domain [MAFA13].

The purpose of an SRA is to contain the vision or the case scenario that is being modeled, which are laboratory devices in this thesis' case [CMV+09]. The domain and vision guide and place the team on the same page to answer the different dimensions of an SRA. The thesis will show later on how these dimensions can help in setting the SRA domain and the high-level architecture of the evaluation process. The dimensions are explained in detail in Angelov et al. [AGG12] and Chauhan et al. [CBS17], but this thesis will briefly summarize them since they will be used to set the design scope of the SRA's goal:

- **SRA Classification Dimension:** This dimension is used to classify the SRA. It answers the following questions:

  - **“Where will it be used?”**: Answers the question about the intended type of organization [AGG12] [CBS17].
  - **“Who defines it?”**: Answers the question if the SRA is intended for a single or multiple organizations [AGG12] [CBS17].
  - **“When is it defined?”**: Answers the question if the SRA is being implemented before or after the development of the concrete software architecture and application [AGG12].
  - **"What is the maturity stage of the domain?":** Checks whether the domain is mature to ensure that an SRA can be designed without a lot of changes in the future [CBS17].
- **SRA Design Dimension:** This dimension sets how the SRA will be used and operated [AGG12]. It also includes how the SRA is detailed, described, and represented [AGG12] [CBS17]. It answers the following questions:
  - **"What is described?"**: Explains what the SRA will include [CBS17]. It can also list the basic elements at a high level [AGG12].
  - **"How is it represented?"**: Describes if the SRA is going to be represented in an informal, semi-formal, or formal manner [AGG12] [CBS17].
  - **"How is it described?"**: Specifies the format (textural, graphical, ...) on how the SRA is described [CBS17].

4 Architecture Evaluation

An architecture evaluation is a process or a framework to guide the design of a software architecture and to provide reasons for following certain architectural decisions. It is used to explain the architecture, check whether adequate decisions and tradeoffs were made, and provide decision points and formal agreements among stakeholders [RW12].

The scope of an SRA in this paper is focused on applying an SRA on a single organization that acts as a facilitator (not a standard) to guide and simplify the design of concrete architectures that are under the same domain as the SRA's. Moreover, the SRA will not include deployable hardware components but rather the common software components and their interactions between them. The SRA's scope is defined to limit the different possibilities of evaluation methods and approaches that can be followed in this thesis and to help better define the SRA's abstraction layer.

4.1 Architecture Evaluation Approach

An architecture evaluation can follow five different, or a combination of several, approaches to evaluate an architecture:

- Previous experience of developers and stakeholders [PS15a].
- A mathematical model to evaluate quantitative quality attributes such as performance, latency, and reliability [PS15a].
- A simulation approach to mimic architectural components at a high level [PS15a].
- A prototype system as a proof-of-concept [RW12].
- A scenario-based approach to identify scenarios in order to analyze their effects on quality attributes [PS15a] [RW12].

Using previous experience as an evaluation approach can be difficult to assess its credibility because experience varies between each and every person. Furthermore, it is impossible to use it if previous experience is not present. As for the mathematical model, it is an appropriate approach to use when quantitative quality attributes are of high priority. However, this paper assumes that the SRA will not include hardware component details which makes the mathematical model an inadequate approach.

As for the simulation and prototype approaches, they can be used as a proof-of-concept to evaluate the usability of the SRA by designing and implementing prototype concrete architecture based on the SRA. The scenario-based approach captures the different quality attributes and scenarios from stakeholders and provides reasons for following certain architectural decisions. The scenario-based approach will be used in this paper because it evaluates the SRA's quality attributes and assesses

how the SRA copes with different scenarios in a particulate domain. While the prototype approach can be used to evaluate the usability of the SRA on how much it facilitates the design of multiple concrete architectures.

4.2 Architecture Evaluation Methods

Several scenario evaluation methods have been proposed, mainly to evaluate software architectures. However, there are not many architecture evaluation methods specific towards SRAs. As a result, this paper determines an evaluation method that is more oriented and focused on the evaluation of SRAs. The evaluation method will be based on architectural comparison according to how well the SRAs satisfy different system architectural drivers.

To determine an evaluation method for SRAs, previous evaluation approaches need to be studied whether they can be taken as a potential source. The evaluation methods were taken from [BZJ], [PS15b], [MGM06], [RZRK19], and [SS12]. Table A.1 in Appendix A contains the evaluation methods that may be used as a source to determine an evaluation method for SRA. On the other hand, table Table A.2 contains the evaluation methods that cannot be used for reaching an evaluation method for SRA.

ATAM and Software Architecture Analysis Method (SAAM) are the most common used evaluation methods and they are both referenced by more than 800 papers each. ATAM is an improvement on SAAM and it is a process that describes how an architecture can be evaluated based on captured quality attributes and scenarios from stakeholders. The evaluation determines risks, sensitivities, and tradeoffs that each architectural decision introduces according to a specific quality attribute or scenario. ATAM is a mature method that provides a clear structure on how to evaluate architecture and is widely used by many different organizations. Out of all the applicable evaluation methods that are mentioned in Table A.1, ATAM will be used as a base to adapt it for evaluating SRA instead of concrete software architecture.

4.3 ATAM Problems for Software Reference Architectures

ATAM is a mature well documented evaluation method that gives a concrete structure to follow in order to evaluate and give reasons behind making certain architectural decisions. In this section, the disadvantages are mentioned and then an extended ATAM method will be presented to address the disadvantages. The drawbacks are based on applying ATAM to evaluate an SRA that its scope was defined at Chapter 4:

- **Drawback 1**: Obtaining quality attributes and scenarios for an SRA is not specified [ATG14] [ATG08]. Stakeholders are used to provide functionalities, quality attributes, and scenarios based on a specific system. This can be challenging for stakeholders if they consider every available and future context the SRA can be applied. It would lead to scenario duplication and an unwanted increase in scenarios that are applicable to one system [ATG14] [ATG08].

- **Drawback 2**: It does not provide a clear method to evaluate quality attributes for a reference architecture [ATG14] [ATG08]. An SRA is at a higher level of abstraction compared to concrete architecture, and ATAM is used to define the low-level details of each quality attribute as a form of scenarios. For example, for the performance quality attribute, there might be a scenario where the run-time threshold must not be exceeded by the system.
- **Drawback 3**: The number of stakeholders can increase a lot if the SRA will be applied to different organizations in the same domain [ATG14] [ATG08] [GVV05]. ATAM suggests to involve as many stakeholders as possible. However, it can be impossible to involve stakeholders from different organizations to list and prioritize quality attributes and scenarios [ATG14] [ATG08] [GVV05]. This issue can still be seen if the SRA is going to be applied to a single organization but stakeholders from different organizations can be interviewed (Example, customers and partners from different organizations that work in the SRA's domain).
- **Drawback 4**: ATAM doesn't support the comparison of architectures against each other to reach the optimal one [BRST05]. ATAM is designed to evaluate the tradeoffs of a single architecture without the consideration of other architecture methods.

4.4 Architecture Evaluation Method for Software Reference Architecture

The evaluation method will be used to answer **RQ1** in Section 1.2. More precisely, it is trying to answer the following question to eliminate ATAM's disadvantages when applying it to SRAs:

- **Question 1**: How to obtain quality attributes and scenarios for an SRA?
- **Question 2**: What procedure can be followed to evaluate the SRA quality attributes and scenarios?
- **Question 3**: If an SRA is applied to multiple organizations, how can stakeholders from one organization represent the other organizations?
  - Is it possible to unite stakeholders from multiple organizations without duplicating quality attributes and scenarios between the organizations?
- **Question 4**: How can different SRAs be compared against each other based on the quality attributes and scenarios?

In this thesis, ATAM is taken as a base and its disadvantages are addressed based on different evaluation methods mentioned in Table A.1 and based on the paper's suggested solutions. In the following sections, ATAM's steps are summarized, discussed, and modified, if needed, to find an evaluation method for SRAs. ATAM's steps are not strict to be followed in a waterfall approach. They can be followed in different order and on several reiterations [KKC00]. In the steps, an architect is mentioned to perform some tasks. The term architect is not strict to an architect person. The architect can be a software team/person, a software evaluator, or whichever person that is running the evaluation process. For simplicity, the term architect is mentioned throughout the paper.

ATAM consists of nine steps:

- **Step 1**: Present the ATAM
- **Step 2**: Present Business Drivers
- **Step 3**: Present Architecture
- **Step 4**: Identify Architecture Approaches
- **Step 5**: Generate Quality Attribute Utility
- **Step 6**: Analyze Architecture Approaches
- **Step 7**: Brainstorm and Prioritize Scenarios
- **Step 8**: Analyze Architecture Approaches
- **Step 9**: Present Results

4.4.1 Step 1: Present the ATAM

ATAM's Summary

This step presents briefly the evaluation process to a small team of stakeholders. A large team of stakeholders will be identified in **Step 2**. The presentation will also include the techniques that will be used to analyze and evaluate architectures and the final output of the evaluation process. The purpose of this step is to answer any stakeholders' questions and to ensure that everyone knows their responsibilities [KKC00].

Discussion

A similar approach to ATAM's can be followed since the step is simple and achieves its purpose of informing the initial small team of stakeholders about the evaluation process and the next steps.

Modification

No modifications are made to this step.

4.4.2 Step 2: Present Business Drivers

ATAM's Summary

A high-level picture of the system will be presented to the stakeholders. The system needs to be understood by all participants. For example, the presentation should include the high level requirements, business goals and constraints, technical constraints, and major stakeholders to be involved in the upcoming steps (the larger stakeholder team) [KKC00].

Discussion

ATAM focuses in this step in presenting the system business drivers. It doesn't go into details about defining the domain which will set boundaries to better define the SRA and to help in comparing different architectural approaches [LBK97]. A well-defined domain will eliminate revalidation and reinvention of already solved solutions [CMV+09] [AGG12]. The architect and stakeholders must find a balanced point to define the domain. If the domain is too specific, then the SRA will be limited and difficult to apply to similar systems. Similar problems can be seen if the domain is too generic where it doesn't capture any valuable information.

Furthermore, ATAM's method is rather focused towards capturing and achieving goals that are related to a system. Additional information can be provided to better explain the goal behind an SRA.

Modification

Some additions need to be done to this step since an SRA depends heavily on the defined domain. The domain should be discussed with the small stakeholder team. It is best specified based on the architect's and stakeholders' knowledge and based on the previous experience of developed systems in that domain. A good way to specify the domain is to set the different dimensions of an SRA. The dimensions were mentioned at the end of Section 3.3.

The second modification is to better understand and present SRA's goals and to identify the goal behind following a specific architectural evaluation process. Examples for such goals are to build a new system, extend an already existing system, assess system's risks, predicate maintenance costs, or reach the most appropriate architecture [KKC00] [LBVB00] [Dol01]. The architect should discuss the goal with the stakeholders since it helps define the main purpose behind a facilitator SRA. Potential goals could be:

- Improve time to market for new systems in the SRA's domain.
- Ease future design of concrete architectures.
- Help non-technical members better understand the systems' domain.
- Describe the desired common quality attributes.
- Capture the common requirements to avoid revalidation and reinvention.

4.4.3 Step 3: Present Architecture

ATAM's Summary

The architect will present a high-level architecture at an appropriate level of detail. The level of detail depends on several factors such as the available information and time. The high-level architecture will contain information such as the different system interactions, technical constraints, and the important use-case scenarios. This step is important as it affects the quality of the analysis [KKC00].

Discussion

This step shows the current available architectural information which will affect the captured scenarios and analysis in the upcoming evaluation steps. However, providing a high-level architecture of an SRA can be confusing since the SRA is already at a higher level of abstraction compared to a concrete architecture. As a result, further information about an appropriate level of detail needs to be provided.

What is important for SRAs are the common requirements for a particular domain (not the requirements that are applicable to a particular system). ATAM does not describe how the system requirements are going to be captured. Stakeholders are most likely used to capture requirements about a certain system and not a domain. Common requirements are at a higher level of abstraction than normal system requirements. They are for example:

- The different user access points (remote, locally, ...) to a system in a domain
- Common business models
- Cloud integration to the system
- Automation

A method to capture the common requirements needs to be specified to ease the process.

Modification

The high-level architecture, should not be detailed. It is more of a diagram showing the overall picture of the domain that was defined in **Step 2**. It should show where the SRA is going to be used and which common system information it will describe. If certain information such as technical constraints or different system interactions are known, then they should be shown in the high-level architecture.

As for common requirements, this paper suggests representing them in a tabular fashion. The idea of the table was taken from Family Architecture Assessment Method (FAAM) and Scenario based Architecture Level Usability Analysis (SALUTA) [Dol01] [FGB04]. The table contains the roles, their common requirements, and requirement type. The requirement type is about determining whether the common requirement is core or additional. Core requirements are requirements that must be available for each system in the specified domain. They must be taken from the SRA. As for the additional requirements, they are requirements that can be added to the system that is being developed in the specified domain. They can be either taken or not from the SRA. Further details about the common requirements will be captured in **Step 5** when capturing the scenarios of each. An example common requirement table is shown in Table 4.1.

Table 4.1: Example Common Requirements Table

Roles	Common Requirements	Requirement Type
Admin, End-user	The device execution can be automated remotely	Core
Admin	System files can be saved and retrieved from the cloud	Additional

Continued on next page

Table 4.1 – continued from previous page

Roles	Common Requirements	Requirement Type
End-user	System must provide undo-redo capability	Additional
End-user	Log-in capability	Core

The purpose of the table is different from FAAM and SALUTA. Initially, it will be used by the architect to present the common requirements to the stakeholders to shape their mindset to focus on thinking about common requirements that can be in the domain rather than a specific system requirement. The presented architect's requirements may be false, but their main purpose is to act as a starting point to let the stakeholders brainstorm the common requirements. Meetings with stakeholders can be repeated on multiple sessions until all of the common requirements are captured in the table and the stakeholders are satisfied with the results.

Additional benefits to obtain common requirements can be seen when stakeholders have previous knowledge and experience on systems that are under the domain. If the stakeholders are still having difficulties obtaining common requirements, then the architect can take some of their existing system specific requirements and move them to a higher level of abstraction during the meeting to let the stakeholders better understand the goal. Furthermore, setting the common requirements with the stakeholders helps in answering parts of **Question 1** since obtaining the quality attributes and scenarios of an SRA is going to be based upon the common requirements.

4.4.4 Step 4: Identify Architectural Approaches

ATAM's Summary

The architect will identify different architectural approaches that can be used. The architect will only identify and not analyze them. It is important here to not disregard any architectural approach because they will be later assessed. The architecture defines the structure of the system and how it can cope with changes and integrate with other systems. The architecture is going to address the requirements and the highest priority quality attributes and scenarios [KKC00].

Discussion

This step will be performed as how it is intended by ATAM because it fulfills the need of identifying the different architectural approaches that can be followed to achieve the SRA. However, the common requirements that are captured in **Step 3** need to be further detailed to identify more suitable architectural approaches. ATAM suggests to identify architecture approaches before getting the quality attributes and the scenarios. However, other evaluation methods do the opposite [LBVB00] [BRST05] [Dol01] [FGB04] [LBK97].

Modification

This paper suggests to perform this step after **Step 5** because generating a quality attribute utility tree helps better refine and detail the common requirements that the SRA should deliver, which leads to better architectural approaches.

4.4.5 Step 5: Generate Quality Attribute Utility Tree

ATAM's Summary

This step is about identifying the quality attributes and scenarios to generate a quality attribute utility tree. Quality attributes and scenarios have been defined in Section 3.1.2 and Section 3.1.2. The utility tree is a tree diagram that breaks the quality attributes into finer details until it reaches a scenario that the architecture should cover [KKC00]. The scenarios will be prioritized according to their importance and how easily they can be achieved [KKC00]. The utility tree is used to guide the team during analyzes when the risks, tradeoffs, and sensitivity points need to be captured [KKC00]. A list of the most used quality attributes is available in the standard ISO/IEC 25010 [ISO11] and in Arvanitou et al. [AAC+17]. From the list, ideas of what quality attributes to include can be taken. Some of the standard quality attributes are maintainability, modifiability, testability, analyzability, stability, changeability, reusability, and comprehensibility [ISO11].

Discussion

As previously mentioned in **Drawback 1**, identifying generic common scenarios for a domain is difficult. Angelov et al. [ATG14] suggests a solution to this by identifying the different contexts that the SRA can be applied to. A utility tree is then obtained for each context from the different stakeholders in multiple organizations [ATG14]. Finally, the utility trees are merged together to obtain the common quality attributes and scenarios for the SRA [ATG14]. This approach is time consuming and causes the number of stakeholders to increase a lot, which was explained in **Drawback 3**, and the approach assumes that the SRA will be applied to multiple organizations which is not always the case.

Furthermore, ATAM suggests that the architect generates the utility tree first. Then the scenarios are brainstormed in **Step 7** with the other stakeholders to capture and prioritize further scenarios. Then the utility tree is updated from all the different sources [KKC00]. For evaluating SRAs, an architect can have difficulties generating a utility tree on his/her own because of the wide range of contexts that a domain can cover. On the other hand, domain experts have a clearer vision about what should be included in the domain.

Modification

Software Architecture Analysis for Evolution and Reusability (SAAMER) generates scenarios based on the objectives that were captured from the stakeholders [LBK97]. Scenarios are generated until domain experts and stakeholders are satisfied that the scenarios cover the objectives well [LBK97]. A similar approach is followed in this paper to obtain common quality attributes and scenarios

based on the common requirements identified in **Step 3**. In addition, the paper suggests following the two approaches that are described in Architecture-Level Modifiability Analysis (ALMA) to obtain the common scenarios. They are used because they help stakeholders generate the utility tree from the only two possible perspectives: a top-down and a bottom-up approach [LBVB00].

- **Top-down Approach**
  - Mentions the quality attributes first (or a subset of the quality attribute) and then identifies scenarios, with the stakeholders, that are related to the quality attributes [LBVB00].
- **Bottom-up Approach**
  - Opposite to the top-down approach. It first lets the stakeholders mention scenarios and then map the scenarios to their respective quality attribute [LBVB00]. If a scenario can be mapped to more than one quality attribute, then the scenario should be divided accordingly so that it only fits into one quality attribute.

The top-down and bottom-up approaches merge **Step 5** (generating a utility tree) and **Step 7** (brainstorm and prioritize scenarios) together. Similar to **Step 3**, capturing common quality attributes and scenarios can be performed on several meetings until the stakeholders are satisfied with the utility tree. Example of a brief utility tree is shown in Figure 4.1. It can be compared with Table 4.1 to see how the common requirements for automation and protocol execution is further defined in the utility tree.

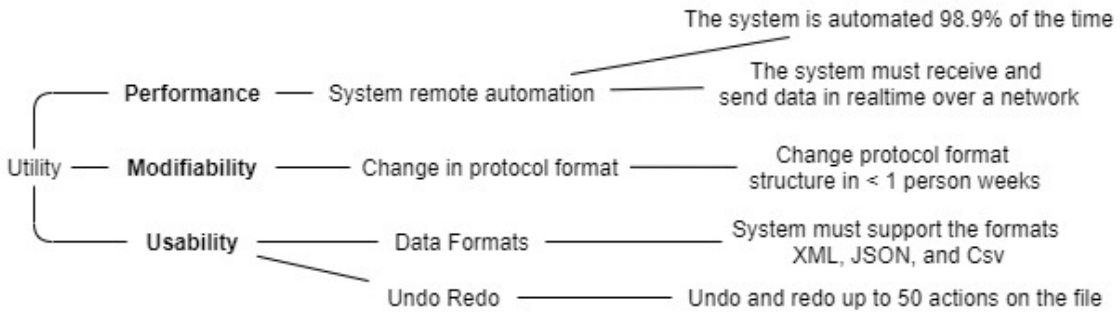

Figure 4.1: A Brief Utility Tree Example

ATAM's utility tree serves to concretize the scenarios [KKC00]. However, getting a concrete value for common scenarios can be difficult. The stakeholders should try their best to refine the scenarios until they reach a concretized value. By following **Step 3** and this step, **Question 1** is answered since they give a structure to follow to obtain the requirements, quality attributes, and scenarios of an SRA.

After constructing the utility tree, the scenarios (leaf nodes) are weighted by all the stakeholders. Weighting scenarios will tell the architect the important quality attributes which will be used later during the analysis step (**Step 5**) to obtain the score of each architecture. They are weighted according to importance. Since prioritizing can be understood differently between stakeholders and it is sometimes difficult to prioritize some scenarios over others [KKC00]. Thus, the prioritization will be based on three levels: High, medium, and low.

4.4.6 Step 6: Analyze Architectural Approaches

ATAM's Summary

This step is about getting sufficient information about the different architectural approaches to assess them. The output of this step is to provide a reason about decisions and to help reach the requirements and quality attributes of the final system. First, a scenario from the utility tree is taken to identify the risks, sensitivities, and tradeoffs for each architectural decision. They are identified by asking questions about how the approaches are related to the quality attributes. These questions are based on the software team experience, software academic material, and documented experience. Then the captured sensitivity points and tradeoffs are categorized as either risks or non-risks. At the end, the team should have a list of risks and an idea about the important aspects of the architecture and which approaches should be taken over the others [KKC00].

Discussion

ATAM's analysis approach is an appropriate method to identify differences between the architectural approaches. However, ATAM focuses on analyzing a single system architecture at a time [BRST05]. Since a facilitator SRA is the goal, it is better to also compare different whole architectures together with comparing different architectural approaches. The whole architecture specifies how the different architectural components can affect each other. Examples of architectural comparison methods are ALMA, SALUTA, and Domain-Specific Software Architecture Comparison Model (DoSAM). ALMA requires to list the affected components to evaluate the architectures modifiability [LBVB00]. SALUTA and DoSAM suggest to give a score for each architecture based on how much they fulfill the quality attributes if the goal is to compare different architectural candidates [FGB04] [BRST05].

Furthermore, high level of abstraction is considered for SRA evaluation. Thus, some quality attributes are less likely to be important or difficult to analyze them for an SRA. For example, the usability and functional suitability quality attributes depend on the concrete implementation rather than on the architecture itself.

Modification

A weighted score method is introduced to compare different SRAs and the architectural approaches that were initially identified in **Step 4**. This step will help the architect reach the most suitable SRA that can fulfill the quality attributes and scenarios that were discovered together with the stakeholders. It is divided into seven sub-steps:

- **Analysis Step 1:** In this step, tradeoffs, risks, and sensitivity points are obtained for each architectural approach that supports its corresponding scenarios. This is done by following ATAM's way. The architectural approaches were initially identified in **Step 4**. In addition, the architecture components that either fulfill or cause risks for the scenarios should be noted [LBVB00]. This will help in identifying the components that need to be further analyzed when SRAs are being designed in **Analysis Step 3**.

- **Analysis Step 2:** Based on the obtained tradeoffs, risks and sensitivity points, the architect will give a score from 1 to 5 on how much each architectural approach supports its corresponding scenarios [FGB04]. If the architectural approach supports the scenario but with high tradeoffs, risks, and sensitivity points then the score is low, and vice versa. Scoring from 1 to 5 was chosen for simplicity. The idea of scoring scenarios is taken from SALUTA [FGB04]. SALUTA sets a score for each scenario on how much the architecture or system supports it. This scoring procedure gives a structure to answer **Question 2** on how to evaluate SRA's quality attributes and scenarios. The procedure is mainly dependent on the architect's experience and knowledge in identifying tradeoffs, risks, and sensitivity points, and scoring them.
- **Analysis Step 3:** Once the score of how much each architectural approach supports its corresponding scenarios is set, the score has to be multiplied with the scenario's priority to get the weight. The architectural approach with the highest weight will be chosen to construct different SRAs based on the different architectural patterns. For example, one SRA can be designed using a client server architectural pattern while another SRA can be designed using a server cluster architecture pattern. The highest weighted architectural approaches should be included in the SRAs. For example, the type of database can be an architectural approach that will be shown in the SRAs.
- **Analysis Step 4:** After constructing different SRAs, the architect scores from 1 to 5 how each overall SRA satisfies each scenario. The entire architecture is considered when scoring because some scenarios' measurability depends on how multiple components function and interact with each other. Similar to **Analysis Step 3**, once the score of each SRA for each scenario is set, the score has to be multiplied with the scenario's priority to get the weight. Then all of the scenario weights that are under the same quality attribute are added together to get the final weighted score of the architecture with respect to each quality attribute.
- **Analysis Step 5:** This step tries to achieve refined SRAs based on the obtained quality attributes' weighted scores. For each quality attribute, concepts and components from the SRA with the highest weighted score should be taken. The architect should try to mix, remove, add, or combine components and architectural decisions to achieve new possible SRAs.
- **Analysis Step 6:** Repeat **Analysis Step 4** but for the new SRAs to identify whether they will get a better weighted score. This step can be repeated several times until no further SRAs can be designed or if the new SRAs have a lower weighted score than the previous ones.
- **Analysis Step 7:** Pick the SRA with the highest weighted score.

This approach gives a framework that can be followed to answer **Question 4**. It provides a weighted scoring method that allows to compare SRAs and to give reason for choosing an SRA over the other options.

4.4.7 Step 7: Brainstorm and Prioritize Scenarios

ATAM's Summary

This step is when stakeholders meet to brainstorm and identify scenarios. No ideas or opinions are disregarded in this step. The identified scenarios are use-case, growth, and exploratory scenarios. Use case scenarios are scenarios where the end-user will interact with the system. Growth scenarios are modifications that might happen to the architecture or system. Exploratory scenarios are unusual scenarios that are used to push the system to its limits. They can be of the form of major modifications to the system or unusual high loads and stress tests. After the scenarios are identified, they are prioritized by the stakeholders. Then the prioritized scenarios are compared with the utility tree to see if they match. Any differences must be reconciled and explained. Finally, the initial utility tree from **Step 5** and the identified scenarios in this step are merged together to obtain a single utility tree [KKC00].

Discussion

ATAM relies on the architect to generate a utility tree in **Step 5**. Afterwards, the identified scenarios from the stakeholders in this step are merged into the utility tree [KKC00]. For an SRA, an architect can find difficulty in identifying most of the scenarios of a particular domain. Moreover, different stakeholders can have different scenarios for the same domain

Modification

This step was merged with **Step 5** to avoid unwanted analysis for architectural approaches since not all scenarios would have been captured yet. For an SRA it is better to identify the utility tree with the stakeholders before identifying different architectural approaches.

4.4.8 Step 8: Analyze Architectural Approaches

ATAM's Summary

This step reiterates **Step 6** if new scenarios have been identified in **Step 7**. If no changes were made to the utility tree in **Step 7**, then this step is a testing activity to uncover any new information. If new information is discovered, then the team should go back to **Step 4** [KKC00].

Discussion

This is the same step as **Step 6**. According to ATAM, all of the steps can be done in an iterative manner in different order.

Modification

The modifications to this step have been previously explained in **Step 6**.

4.4.9 Step 9: Present Results

ATAM's Summary

The final step is to document the results to give reasons on why a specific architecture approach was chosen over the others. ATAM suggests to present everything to the stakeholders but the most important thing is the outputs of ATAM such as the utility tree and the architectural analyses [KKC00].

Discussion

Documenting should be based on the goal of the analysis and the captured requirements [FGB04]. Furthermore, if previous systems exist in the domain, it would be also beneficial to document how the concrete architecture can be adapted to the SRA.

Modification

No modifications are made to this step.

4.4.10 Steps Summary

Figure 4.2 is a flowchart that summarizes the modified ATAM evaluation process to orient it towards SRAs. The steps are not in strict order and the flow can go back if previous steps need more clarification or if new information has been identified.

The first step is to present the evaluation process to the initial small team of stakeholders. This step will be followed the way ATAM proposed with no modifications.

The second step is to present the business drivers to the initial small team of stakeholders and to identify the major stakeholders that will be involved in the next steps. In this step, ATAM does not specify that a domain needs to be defined and focuses on achieving system goals. Therefore, two modifications were made to this step. First modification is to define the domain by answering the dimension questions that were presented at the end of Section 3.3. The second modification was to better understand the SRA's goals by listing some advantages of SRAs. The most notable one is to ease future design of concrete architectures for multiple systems within the domain.

The third step is to present the high-level architecture to major stakeholders. For SRAs, details on how to define a high-level architecture are needed since an SRA is already at a higher level of abstraction compared to concrete architecture. Therefore, the high-level architecture for an SRA is going to be a non-detailed diagram that represents the defined domain. Furthermore, ATAM does not provide a way to capture common requirements. Thus, a tabular fashion was proposed to

capture them. The table will initially be used by the architect to present an initial set of common requirements to shape stakeholders' mindset towards focusing on the domain to capture common requirements.

ATAM suggests to identify architectural approaches in the fourth step and generate a utility tree in the fifth step. This paper suggests performing them in reverse order. First the utility tree will be constructed, and then different architectural approaches will be identified. This helps in better refining the captured common requirements which leads to better initial architectures. Also, further details on how to capture common scenarios and quality attributes for SRAs were provided based on ALMA's approach in identifying scenarios.

The sixth step is to analyze the different SRAs according to the utility tree. A modified analysis method from ATAM's was introduced because ATAM focuses on analyzing a single system architecture at a time. The analysis method consists of seven steps. The first analysis step is to identify tradeoffs, sensitivity points, and risks for each architectural approach. The second analysis step is to score the architectural approaches on how much they support there corresponding scenarios based on the identified tradeoffs, risks, and sensitivity points. The third analysis step gets the highest weighted architectural approaches to design different SRAs based on the different architectural patterns. In the fourth analysis step, the architect scores how much each SRA satisfies each scenario to obtain the overall weighted score of the SRA. The fifth analysis step tries to achieve refined SRAs based on the obtained quality attributes' weighted score by combining, modifying, adding, or removing components from the different SRAs. The sixth analysis step is to repeat the analysis from the fourth analysis step but for the refined SRAs. Finally, in the seventh analysis step, the SRA with the highest weighted score will be chosen.

The seventh and final step of the evaluation process is to document the results and present them to the stakeholders. This step is performed similar to how ATAM suggested.

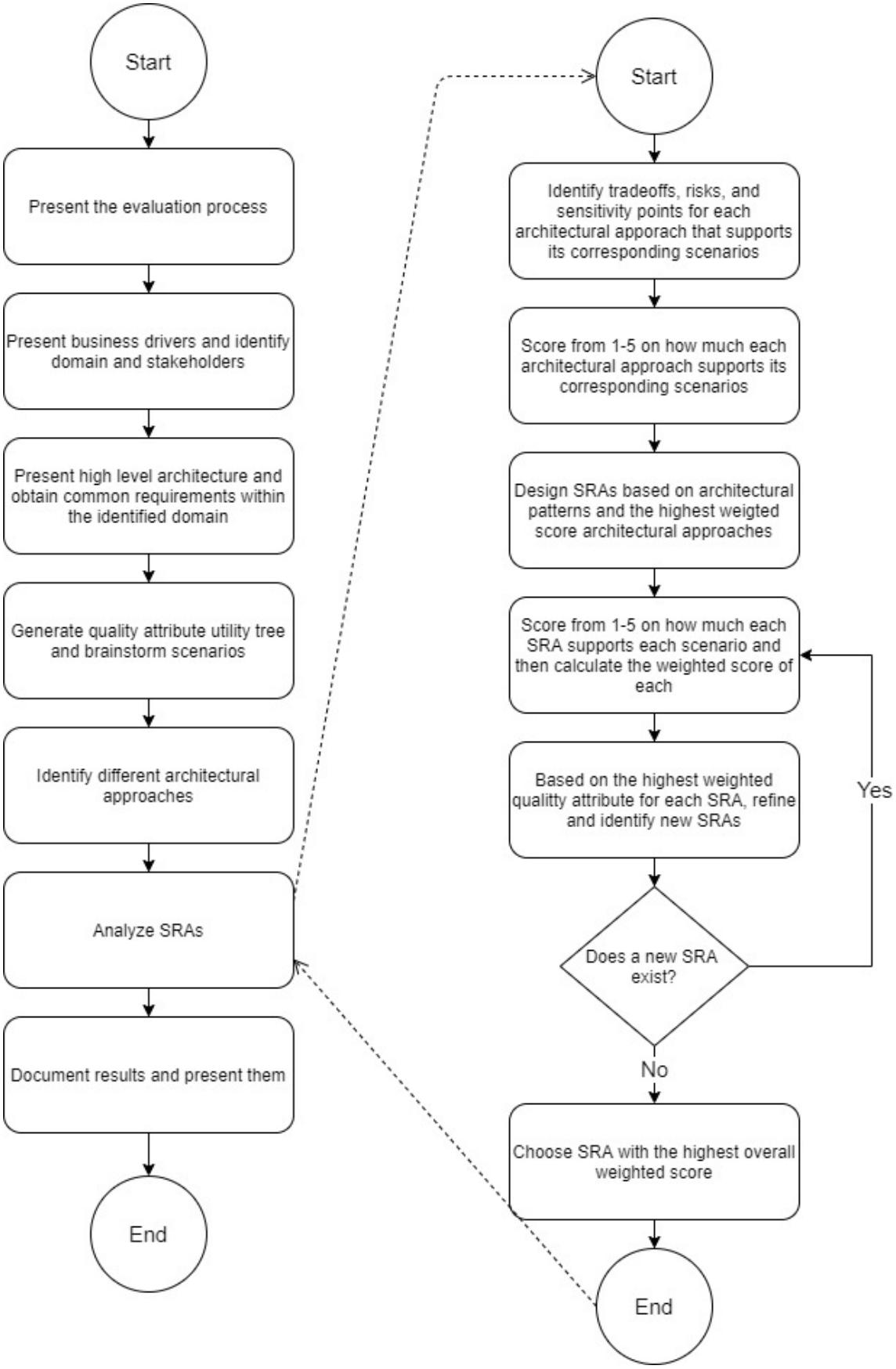

Figure 4.2: Flowchart of the evaluation steps

5 Software Reference Architecture for Laboratory Devices

In this chapter, the paper will present an example where the evaluation method was applied to obtain an SRA for laboratory devices in particular liquid handling devices. The chapter presents how the stakeholders performed each evaluation method and the results of each. It also presents challenges or uncertainties faced, and slight changes to the evaluation proposed by the stakeholders or architect when applying the evaluation steps.

Overall, the proposed evaluation method for SRA was proven to be beneficial because the architect was able to reach an SRA for liquid handling devices based on the stakeholders' needs and requirements. Slight modifications to some evaluation steps were made to ease the process, but this may vary for other stakeholders and architects that are applying the evaluation method.

5.1 Introduce the Evaluation Process

5.1.1 Evaluation Step Results

Software architecture and the problems that were mentioned in Section 1.1 were presented to the initial small team of stakeholders. The main goal of designing a facilitator SRA was then presented together with its benefits to ease the development of liquid handling devices. Then the evaluation process was briefly explained so that each stakeholder will know their responsibilities for the next steps. Finally, stakeholders' questions were answered.

5.1.2 Comments About Evaluation Step

The architect and the initial small team of stakeholders didn't have any difficulties with this step.

5.2 Discuss Business Drivers

In this step the primary stakeholders defined the domain and identified the secondary stakeholders that can be involved in the process for the following steps.

5.2.1 Domain Results

The team has defined the following domain based on the domain dimension questions mentioned at the end of Section 3.3.

SRA Classification Dimension

- **Where will it be used?** The SRA will be used as a facilitator to reach concrete architectures in organizations that develop software for laboratory devices. Since there are a wide variety of laboratory devices, the scope of laboratory devices is going to be limited to liquid handling devices to have a more accurate SRA. The outcome of liquid handling devices could be in the form of a physical, for example mixing liquids, or data output, for example measuring liquid volumes. The user interacts with it by passing protocols as inputs where the device's output is the output of running the protocol. A laboratory device protocol is a set of instructions set by the user. It contains the steps that the device needs to follow to achieve the outcome. The laboratory device protocols are usually created and executed using a software application that is deployed on the device. An example of a laboratory device in this domain is the I.DOT One shown in Figure 5.1. The I.DOT One is a device used to dispense liquids from one laboratory plate to another in micro and nano-liter form. The protocol instructions of this device are the liquid dispensing steps. The application that is used to create and run the protocols is deployed on a tablet that is mounted on the device as shown in Figure 5.1.

  Stakeholders also specified the device's price range. Liquid handling devices can have different requirements and scenarios based on the different devices' price ranges.
- **Who defines it?** The SRA is intended for a single organization.
- **When is it defined?** Since the organization already has systems that are built in that domain, the SRA is going to be defined after the development of concrete software architecture.
- **What is the maturity stage of the domain?** The domain is quite mature but new ideas about laboratory device communication protocols, standards, cloud connectivity, and automation are still under discussion and are implemented differently between organizations.

Figure 5.1: I.DOT One Laboratory Device

SRA Design Dimension

- **What is described?** The SRA should describe the system elements and the relations between them. It will also include the system properties and communication protocols that should be followed, if any.
- **How is it represented?** The SRA will be represented in a semi-formal way.
- **How is it described?** The SRA will be described by a mix of graphical and textural notations. The graphical notation, Unified Modeling Language (UML) like notation in this case, will be used to represent the key components of the system. The textural notation will explain finer details about the graphical components.

5.2.2 Comments About Domain

This is an improvement over the normal ATAM evaluation process. ATAM only presents the system business drivers and does not specify the domain. While in this case, stakeholders did not have any difficulties identifying the domain and found it easy to follow the dimension questions.

5.2.3 Stakeholder Results

Since the domain is defined for a single organization, stakeholders were identified within the same organization. Stakeholders from different roles were chosen to capture liquid handling devices from different perspectives. They comprised of firmware engineers, software developers, product owners, and application scientists.

5.2.4 Comments About Stakeholder

Identifying stakeholders was simple because the organization is a small to medium scale organization. Email communication was relied on as a consequence to the difficulty of bringing all stakeholders for a meeting due to different time schedules.

5.3 Identify Common Requirements

5.3.1 Evaluation Step Results

Figure 5.2 shows the high level diagram that was presented. It shows the external system components that are going to interact with the laboratory device software. The device is going to be accessed via the software that is deployed on the device, external user personal devices, cloud service to save and share device data, and a laboratory management system for automation.

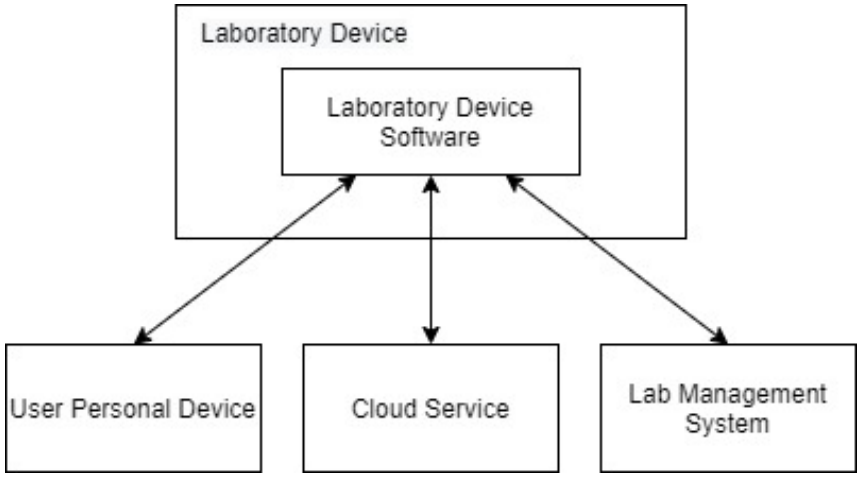

Figure 5.2: High Level Diagram

Table 5.1 shows the captured common requirements for liquid handling devices. The common requirements were captured based on the stakeholders' knowledge about the domain and previous documentations for some parts of the liquid handling devices. First, the common requirements were identified by the architect then they were shared with the stakeholders. Each stakeholder contributed by modifying, suggesting, and adding requirements based on their knowledge.

Table 5.1: Liquid Handling Devices Common Requirements

Roles	**Common Requirements**	**Priority**
Laboratory-User	User can create, view, save, edit, and delete plates.	Core
Laboratory-User	User can create, view, save, edit, and delete liquids.	Core
Laboratory-User	User can create, view, save, edit, and delete device protocols. Device protocols consists of protocol settings, chosen liquids, chosen plates, and protocol steps.	Core
Laboratory-User	User can execute device protocols. Also, use can monitor and interrupt running device protocols. Dispensing depends on the device settings and device protocol.	Core
Laboratory-User	Software displays feedback of the executed device protocol whether it was successful or not.	Core
Laboratory-User	Software displays a detailed result of the execute device protocol.	Additional
Laboratory-User	User can simulate the execution of a device protocol. The simulation will show the protocol steps that the device will follow without executing them.	Core
Laboratory-User	A device protocol cannot be overwritten. A new device protocol needs to be created every time the protocol is saved.	Additional
Laboratory-User	User can undo and redo actions when editing a device protocol.	Additional

Continued on next page

Table 5.1 – continued from previous page

Roles	Common Requirements	Priority
Device	Technical admin can run stress test device protocols that are unsupervised for hours.	Core
Laboratory-User, Laboratory-Manager	System must record all the protocols that have been executed.	Core
Laboratory-User	Lab-Manager must assign a user group to a protocol where only the assigned user group have access to the protocol.	Additional
Laboratory-Manager	Protocols can be assigned to one user where only that user can use the protocol.	Additional
Laboratory-Manager, Technical Admin	Lab-Manager can assign users to a user group. A user group is a collection of users.	Additional
Device	The system must support multiple hierarchical user roles.	Additional
Laboratory-Manager, Technical Admin	Lab-Manager or technical admin can add, modify, and remove device software settings.	Core
Laboratory-User, Technical-Admin	Lab-Manager or technical admin can add, modify, and remove device hardware settings.	Core
Technical-Admin	User or technical admin can calibrate device hardware settings.	Core
Laboratory-User	Admin can update device's firmware.	Core
Laboratory-Manager	User must log-in to use the system and device.	Additional
Laboratory-User	Lab-Manager can create, modify, and remove user accounts.	Additional
Automation System	User can automate the device via a laboratory information management system (LIMS) using the SiLA 2 standard.	Additional
Laboratory-User	Automation system can get the hardware component (instrument) status.	Core
Laboratory-User	User can use the device's system locally (Example, screen that is mounted on the device).	Core
Device	User can use the device's system remotely (Example, personal computer connected to the device via wlan etc.).	Additional
Device	System logs information (Example: user actions) and errors on the device.	Core
Laboratory-User	System logs information (Example: user actions) and errors on the cloud.	Additional
Laboratory-User, Technical-Admin	User can save and share device protocols on and from the cloud.	Additional
Laboratory-User, Technical-Admin	The user can export logs from the software to a specified location.	Additional

Continued on next page

Table 5.1 – continued from previous page

Roles	Common Requirements	Priority
Laboratory-Manager	A built in screenshot mechanism on the software that allows the user to take screenshot of the software because it is hard to take screenshots on the tablet when an action or error occurs.	Additional
Laboratory-User	The Lab-Manager has the ability to archive and unarchive device protocols, liquids, plates, and settings.	Additional
Device	The user can export and import device settings and data to and from other similar devices.	Additional
Device	Logging can be on debug, trace, or production level.	Additional
Device	The system can run as either 21 CFR Part 11 complaint or not.	Additional
Device	Only one user can access the device at a time (No transaction methods are needed when data is being modified).	Core
Device	All of the changes and actions done on device data, software data, and protocols should be logged.	Core

5.3.2 Comments About Evaluation Step

Stakeholders were able to provide valuable common requirements for the domain instead of a particular system. What helped them is their previous experience in the domain, the high level architecture diagram in Figure 5.2, and the initial common requirements presented by the architect. Unlike ATAM this gave stakeholders a structure to start in order to identify common requirements.

Choosing stakeholders from different roles also presented its benefit in capturing all the different functionality aspects. Some stakeholders could not contribute to some requirements because they were not related to their role or knowledge. Thus, stakeholders also suggested to specify if a requirement is specific to a certain stakeholder group. This ensures that the software team or architect will ask the right stakeholder group if additional clarification is needed about a requirement.

5.4 Identify Quality Attributes and Scenarios

5.4.1 Evaluation Step Results

The architect made an initial utility tree based on the common requirements that were previously captured. The utility tree was then verified by the larger stakeholder team that were also identified in the previous step. Stakeholders added, modified, and removed scenarios where needed. No specific order of top-down or bottom-up approach was followed. The approaches were interchangeably applied by the architect and the stakeholders. After the utility tree was constructed, the scenarios were given a priority by each stakeholder individually. If the scenario priority matched with the majority of the stakeholders, then the priority is set. If it did not match, then the scenario should be rediscussed between the stakeholders to identify any concerns.

Table 5.2 shows the final utility tree that was captured. The quality attributes are from the ISO/IEC [ISO11] standard.

Table 5.2: Utility tree

Quality Attribute	**Scenario**	**Priority - High(H) Medium(M) Low(L)**
Compatibility - Interoperability	Device can be operated via SiLA2.	H
Functional Suitability - Functional correctness	The ratio of successfully executed device protocol steps over the failed ones should be greater than 90% on average if hardware dependencies are correct.	H
Maintainability - Modifiability	Adapt a new protocol format in < 1 person week (Ex: adding JSON as a new format).	M
Maintainability - Modifiability	Implement, test, and deploy a new SiLA2 endpoint automation command in < 3 person day.	H
Maintainability - Modifiability	Implement, test, and deploy a new communication protocol endpoints to the system (For example: AutoIT) in < 3 person weeks.	L
Maintainability - Modifiability	Implement, test, and deploy a new parameter to the device settings in < 3 person days.	M
Maintainability - Modifiability	Implement (or modify), test, and deploy a protocol format structure in < 1 person week (Ex: changing plate names, introducing a new plate type).	H
Maintainability - Modifiability	Change log format structure in < 2 person days.	L
Maintainability - Modifiability	Integrate a completely new software UI design in < 10 weeks of work (Ex: Only desktop UI apps are available, and we would like to integrate a new website UI to the system).	L
Maintainability - Modifiability	Implement (or modify), test, and deploy a software end point and logic to a hardware component in <1 person week.	H
Maintainability - Reusability	80% of the architecture should be reusable for liquid lab devices.	H
Performance - Capacity	The entire device log size should not exceed 500 Mbs.	L
Performance - Time behavior	Opening (loading) a protocol locally should not exceed 15 seconds.	H
Performance - Time behavior	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H

Continued on next page

Table 5.2 – continued from previous page

Quality Attribute	**Scenario**	**Priority - High(H) Medium(M) Low(L)**
Performance - Time behavior	The system can be automated by a lab-automation system for a maximum of 77% of the time (The user can use the device for 33% of the time).	H
Performance - Time behavior	The system must receive and send data in real time over a network.	H
Portability - Adaptability	The system should be able to run protocols, config files, and all data related files that are at least 2 versions behind.	M
Security - Accountability	Log all the changes, and by whom they were made, for any protocol at any point in time	H
Security - Accountability	Log all the protocols that were executed with their results and device settings that were used.	H
Security - Accountability	Log all login details (failed/successful attempts, when logged-in and logged-out).	H
Security - Accountability	Log all changes and by whom they were made, to the device hardware and software settings.	H
Security - Confidentiality	Only user accounts with access to automation are allowed to run the automation commands.	H
Security - Confidentiality	The protocol is only accessible to the authorized group specified by the protocol group.	H
Security - Integrity	An unauthorized person should not be able to login to the device and use it.	H
Usability - User interface aesthetics	The size of the undo and redo buffer is at least 50 permutations.	M
Security - Integrity	The protocol can be hashed with an electronic signature which is saved on both the protocol and the DB, and it cannot be copied.	H
Security - Integrity	All data (database information) including log files can be encrypted in production.	H
Maintainability - Modifiability	Backup entire database in <1 person day.	L
Performance - Time behavior	Protocol, liquid, and plate data can be archived and unarchived in less than 15 seconds.	H
Performance - Time behavior	Update device firmware should take less than 3 minutes.	L
Performance - Time behavior	Save and retrieve protocols on and from an external file storage (Ex: cloud) in real time (no noticeable difference between a local and an external file storage).	L

Continued on next page

Table 5.2 – continued from previous page

Quality Attribute	Scenario	Priority - High(H) Medium(M) Low(L)
Maintainability - Modifiability	Change the cloud endpoint immediately from a config file.	M
Maintainability - Reusability	The remote system (that is connected to the device) must be independent from the framework used to build it.	M
Security - Confidentiality	Only users that have access to a protocol, can view the changes that were made to the device protocol and when was the protocol executed with the execution results.	M
Compatibility - Interoperability	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H

5.4.2 Comments About Evaluation Step

Stakeholders had no difficulty in using the top-down and bottom-up approach to identify scenarios. They also prioritized the scenarios as high, medium, or low on how likely the scenario will occur or is needed for liquid handling devices.

Even though there weren't many stakeholders, it was difficult to schedule a meeting due to time schedules overlapping. Thus, email communication was relied on. This resulted in delays as emails had to be sent back and forth to explain the evaluation step. Scheduling a meeting would have been faster and easier as stakeholders can share ideas and thoughts at the same time.

5.5 Define Architectural Approaches

5.5.1 Evaluation Step Results

Based on the identified common scenarios and requirements, the architect proposed different architectural approaches. No approaches were disregarded at this step and none of the approaches were analyzed. Furthermore, the architect found it easier to identify architectural approaches by clustering scenarios and requirements based on whether the cluster is affected by the same approach type. For example, scenarios and requirements that are related to data audit logging were grouped together because the database type affects them. Table 5.3 shows the scenarios that are related to audit logging and the different architectural approaches that can fulfill the scenarios. In the next step, the architectural approaches will be analyzed according to how well they satisfy the scenarios based on the advantages and risks of each. For a complete list of the architectural approaches for the different scenario clusters, please refer to Table A.3 in Appendix A.

Table 5.3: Architectural Approaches for Audit Logging

Scenario	Architectural Approach
Log all the changes, and by whom they were made, for any protocol at any point in time	Log on files (File management system)
Log all the protocols that were executed with their results and device settings that were used	Log on database
Log all changes and by whom they were made, to the device hardware and software settings	
Logs all login details (failed/successful attempts, when logged-in and logged-out)	
Device log size should not exceed 500 Mbs	
Change log format structure in < 2 person days	

5.5.2 Comments About Evaluation Step

The architect suggested to cluster scenarios based on the architectural approach type. If multiple scenarios are affected by an architectural approach, then they should be clustered together. This made it easier for the architect to identify architectural approaches and to score them. Furthermore, this step was done by the architect alone with some feedback from the software team.

5.6 Analyze Architectural Approaches

5.6.1 Evaluation Step Results

The first step was to identify tradeoffs and sensitivity points for each architectural approach according to the scenarios that the architectural approach supports. However, the software team found it easier to think of advantages and disadvantages of each approach. The terms advantages and disadvantages are just easier to grasp for the team compared to the terms tradeoffs and sensitivity points. Fortunately, tradeoffs and sensitivity points can be extracted from the approach's disadvantages. Though it is advisable to identify tradeoffs and sensitivity points as it gives a better distinction between the architectural approaches.

Then the software team ranked from 1 to 5 how much each architectural approach can support its corresponding scenario. The team didn't have any difficulties ranking since they were ranking based on the advantages and disadvantages that they have identified. The highest weighted architectural approaches gave the team a clear idea of what components the potential SRAs should consist of such as the type of database, file formats, and frameworks. For example, Table 5.4 shows the advantages and disadvantages of each architectural approach related to audit logging. While Table 5.5 shows the score and weighted score of each architectural approach with respect to a scenario. From Table 5.5 it is concluded that the system should log audits on a database since it has a higher overall weighted score than logging on files. For the complete list of architectural approaches and weighted scores, please refer to Table A.4 and Table A.5 in Appendix A.

Table 5.4: Architectural Approaches for Log Data Handling Advantages & Disadvantages

Architectural Approach	Advantages	Disadvantages
Log on files (File management system)	The audit log can be easily shared and copied if needed.	Hard to keep track of the log files for multiple device protocols. File has to be locked to ensure that the user cannot change the log manually. A change in the log format structure will result in log files having different format from the previous versions.
Log on database	Can have a structured database that contains the audit log of the device protocols. Log format depends on how the audit log is retrieved from the database.	Requires the device protocols to be saved on the database and not in the file storage. Additional effort is required to retrieve the audit from the database.

Table 5.5: Architectural Approaches for Log Data Handling Weighted Score

Architectural Approach	Scenario	Priority	Score	Weighted Score
Log on files (File management system)	Log all the changes, and by whom they were made, for any protocol at any point in time.	H	1	3
	Log all the protocols that were executed with their results and device settings that were used.	H	3	9
	Log all changes and by whom they were made, to the device hardware and software settings.	H	3	9
	Log all login details (failed/successful attempts, when logged-in and logged-out).	H	3	9
	Device log size should not exceed 500 Mbs.	L	5	5
	Change log format structure in < 2 person days.	L	3	3
			Total	**38**
Log on database	Log all the changes, and by whom they were made, for any protocol at any point in time.	H	4	12

Continued on next page

Table 5.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
	Log all the protocols that were executed with their results and device settings that were used.	H	5	15
	Log all changes and by whom they were made, to the device hardware and software settings.	H	5	15
	Log all login details (failed/successful attempts, when logged-in and logged-out).	H	5	15
	Device log size should not exceed 500 Mbs.	L	5	5
	Change log format structure in < 2 person days.	L	5	5
			Total	**67**

The architect and the software team found it best to use arc42 as a guideline to represent the SRAs into multiple views: deployment, building block, and a database view. From the highest weighted architectural approaches, the architect came up with Figure 5.3 which shows the deployment view with the building blocks for liquid handling devices. The view shows how the users can interact with the liquid handling device, either via the touch screen PC that is mounted on the device, from a remote user PC, or from a laboratory information management system for automation. Also, the application frontend has a connection to an external cloud provider to save, retrieve, and share data. Components that are provided by 3rd party systems are noted as black boxes.

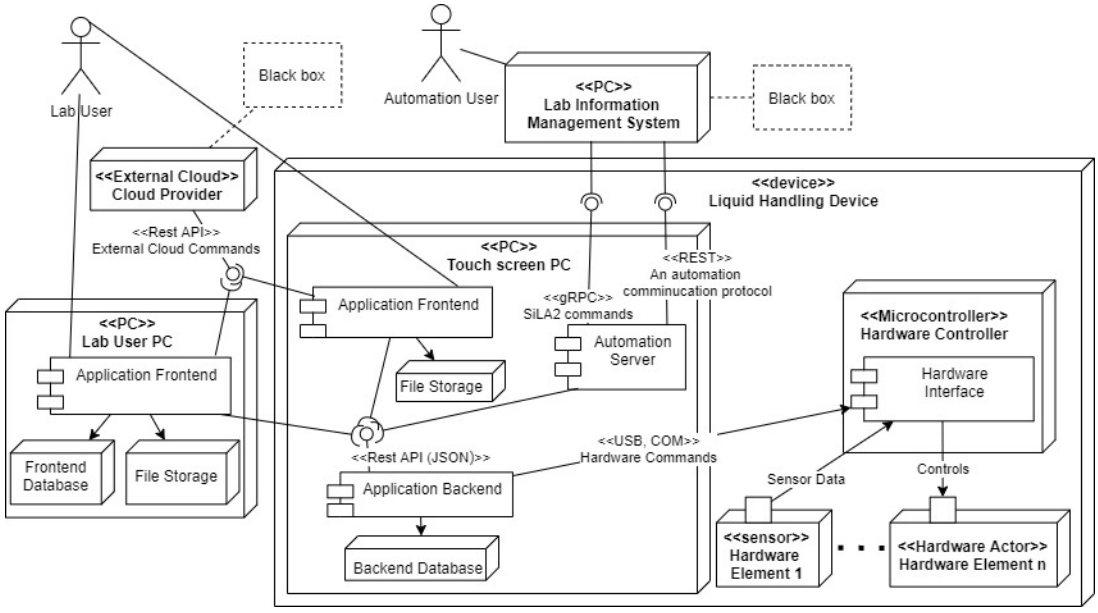

Figure 5.3: Deployment View with Building Blocks for Liquid Handling Devices

As seen in Figure 5.3, there are two databases: *Frontend Database* and *Backend Database*. This is because the architect and the software team had many different ideas on where to host the database. Some suggested that the database can be hosted on the liquid handling device, on a on-premise central machine, or on an off-premise cloud machine. Figure 5.4 shows all the different possibilities on where the database can be hosted. After comparing the different locations, the architect and the software team decided to settle with **Option B** from Figure 5.4 because of the following:

- It is applicable to all costumers independent of their data privacy policies.
- Does not require the costumer to have an IT team to manage the availability of a machine that hosts the database.
- Makes the liquid handling device more appealing to customers since it comes configured straight out of the box without the need of additional hardware to host the database.
- Using the *Frontend Database*, the user can use the liquid handling device's software in an offline mode without having to connect to the device.
- Hosting a database on a separate machine can increase complexity because of the different requirements and privacy concerns that the customer might impose.

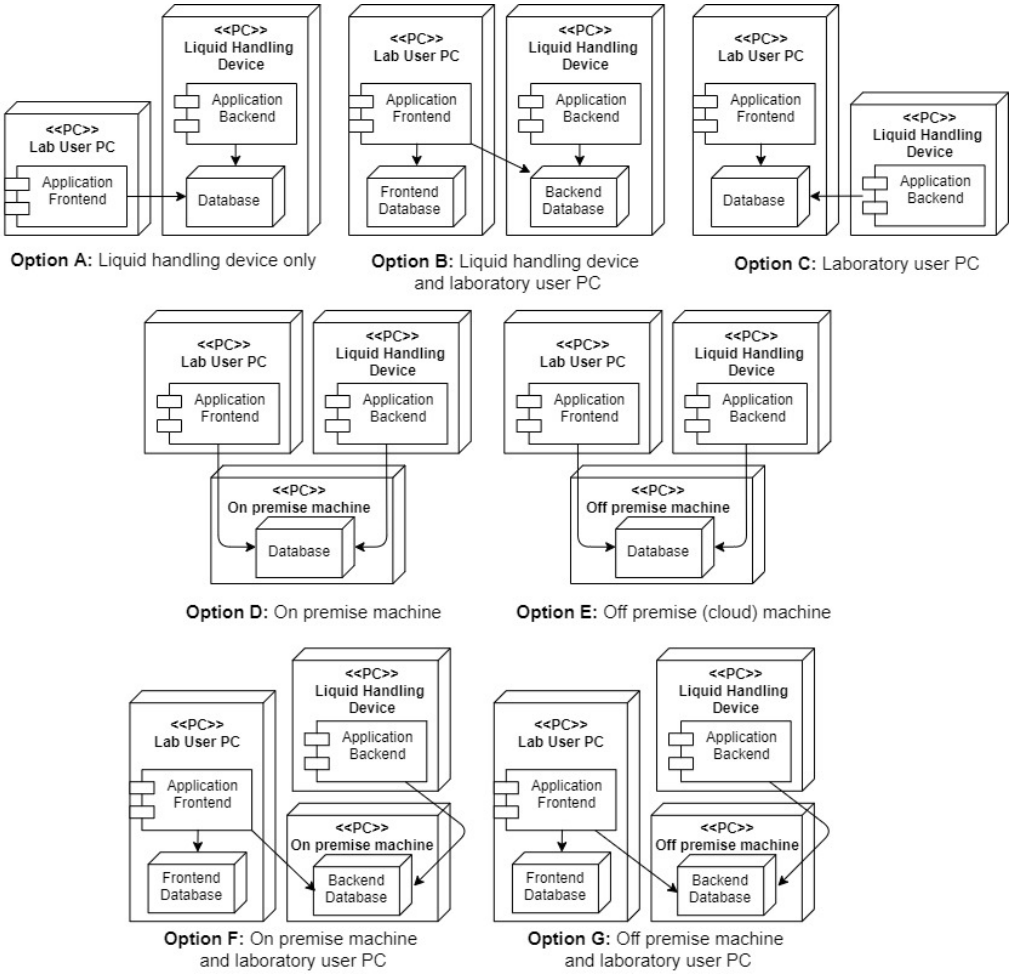

Figure 5.4: Database Host Locations

As for the lower building block views, the architect came up with different SRAs from different architectural patterns. A client server architecture and a message bus architecture shown in Figure 5.5 and Figure 5.6 respectively. The client server architecture uses ZeroMQ as a communication framework between the client and the server, on the other hand, the message broker architecture uses MQTT (HiveMQ) to implement the bus (broker). The software team analyzed both SRAs and scored them from 1 to 5 on how much they satisfy each scenario. Then each score was multiplied with the scenario's priority to obtain the weighted score of each quality attribute and the overall weighted score of the SRA.

The message bus architecture scored slightly higher only in the maintainability - modifiability quality attribute. Thus, the client server SRA in Figure 5.5 was chosen. No new refined SRA approaches were identified because the client server SRA has a higher score on all the other quality attributes and the architect did not identify any possible improvements. For a detailed weighted score difference between the client server and the message bus architecture, please refer to Table A.6 in Appendix A.

If the laboratory users would like to have a centralized machine that can control multiple laboratory devices, then the bus architecture should be followed where both the database and the *Application Backend Bus* are hosted on the centralized machine. Similar to following **Option D** in Figure 5.4 with the message bus architecture where the *Application Backend Bus* will be hosted on the on premises machine.

For the database diagrams, please refer to Figure A.1, Figure A.2, and Figure A.3 in Appendix A.

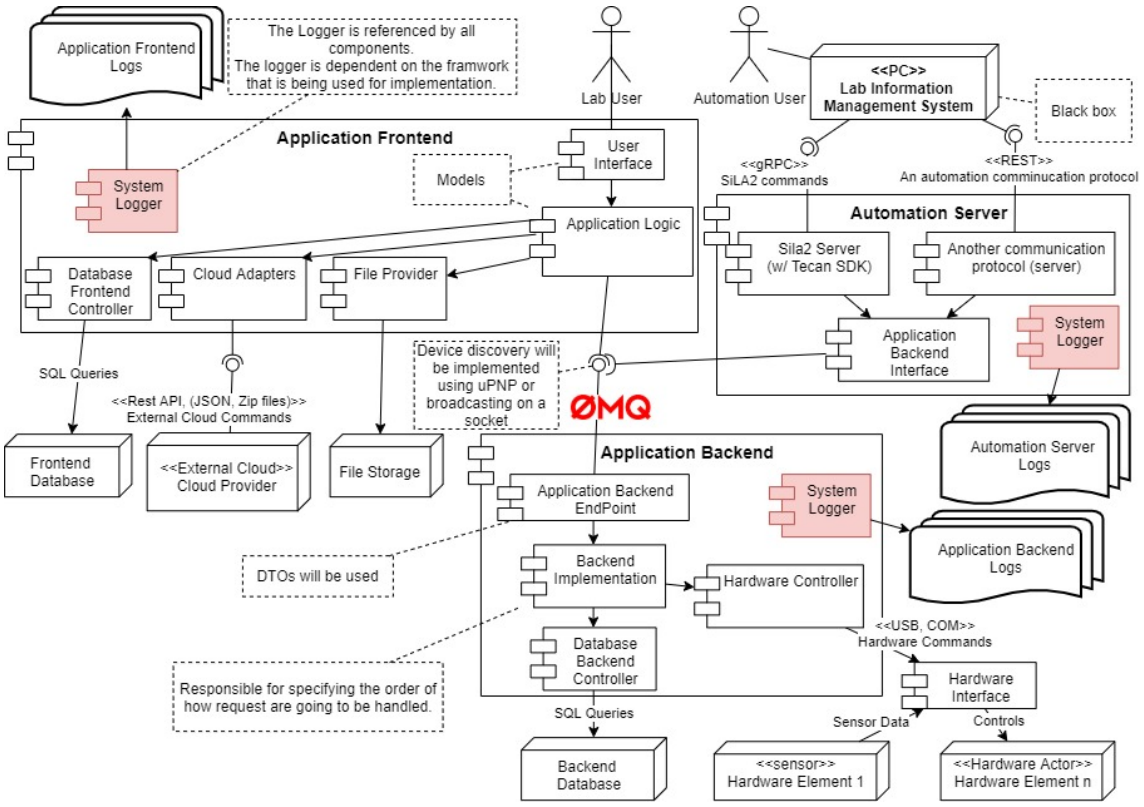

Figure 5.5: Client Server Building Block View for Liquid Handling Devices

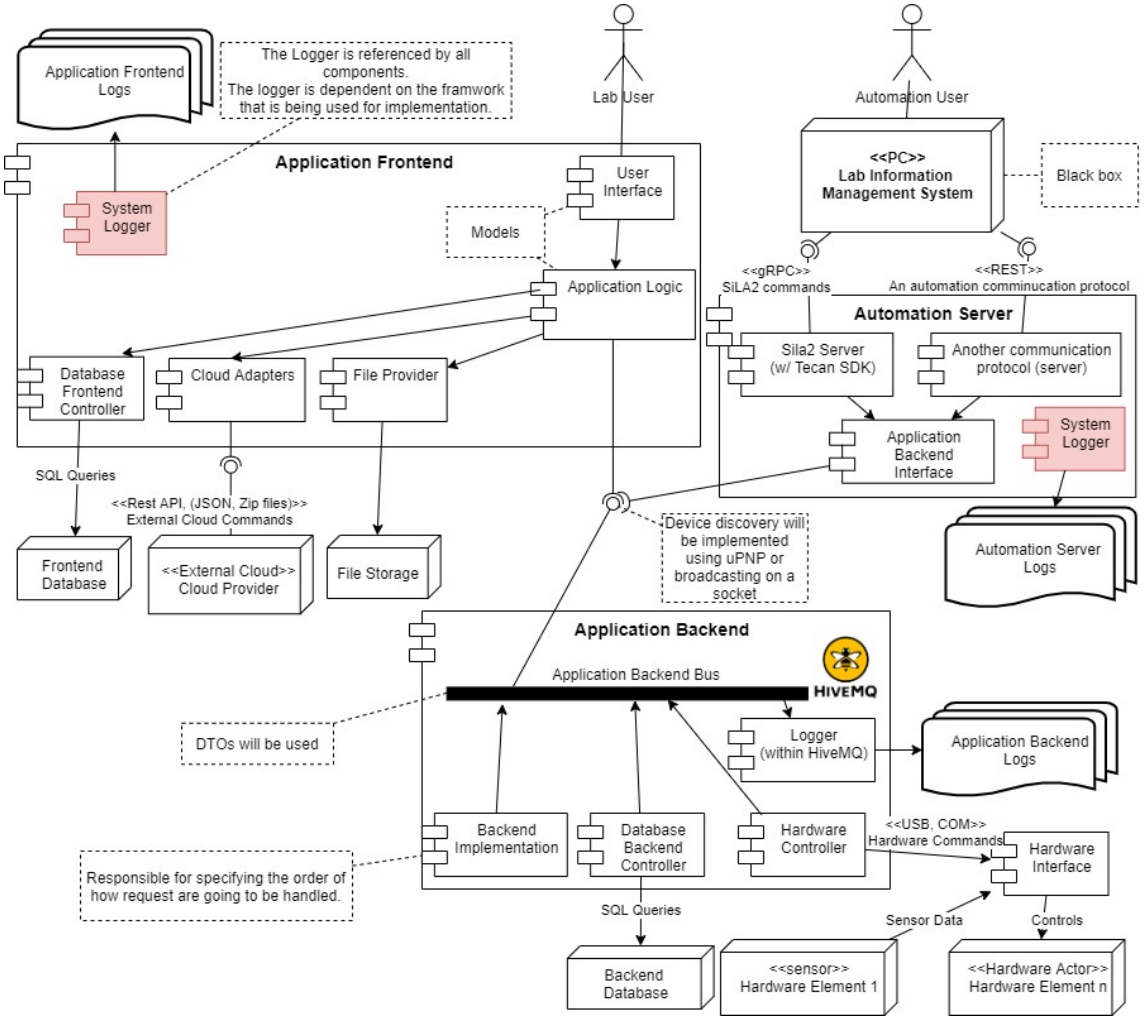

Figure 5.6: Message Bus Building Block View for Liquid Handling Devices

5.6.2 Comments About Evaluation Step

The software team and the architect made several comments when following the analysis steps. The first was that the terms advantages and disadvantage were more convenient than tradeoffs, risks, and sensitivity points when comparing architectural approaches. This may be different with other software teams, but this change of terms didn't cause any problems with the analysis because the main purpose is to score the architectural approaches.

In some cases when scoring architectural approaches, it was not possible to have a single architectural approach as the winner because it can be dependent on the concrete system and the software team that is implementing it. For example, using Object Rational Mapping (ORM) depends on whether the database in the concrete system is required to be interchangeable or not. It also depends on the software team's knowledge in writing data queries and the complexity of the data queries that will be implemented. Another example is the used logging framework. It depends on the programming language and the communication framework (such as Apache Kafka, RabbitMQ, and MQTT) that will be used.

The software team also used arc42 to represent the SRA into a deployment and building block views on different levels. That was based on preference, though other modeling techniques can be used.

It was also hard to score some scenarios especially the ones that depend on the actual system implementation such as performance. Some of the scenarios were given a similar score across all the different SRAs because it is not possible to score them without implementation. For example, the scenario *Opening (loading) a protocol should not exceed 15 seconds* depends on the actual implementation of the system. One could do prototype implementations to measure performance, but that can be time consuming and ineffective especially if the performance scenarios have a low priority. Fortunately, these scenarios are already captured and can be used to test the actual concrete system once implemented.

Overall, the analysis steps gave the architect and the software team a structure that they followed to reach an SRA for liquid handling devices.

5.7 Document Results and Decisions

5.7.1 Evaluation Step Results

All the requirements and scenarios were documented. They can be used by the stakeholders as a starting point if they want to implement a concrete system under the same domain. This will ease the process since most important aspects were already captured. The SRA was also documented which is used by the stakeholders and software team to better understand how the concrete architecture will look like and to provide them a reason why certain software approaches were not taken.

5.7.2 Comments About Evaluation Step

No comments or difficulties were faced in this step.

6 Testing the Liquid Handling Device Architecture

The client server SRA was achieved and evaluated in Chapter 5 using the adapted evaluation method. In this chapter, the paper will test the SRA's effectiveness in facilitating the design and development of concrete systems. The I.DOT One and I.DOT Mini liquid handling devices are used as a case study. Due to the company's privacy, the specific requirements and qualities of I.DOT One and I.DOT Mini are not going to be specified. Instead, the paper will provide a brief discussion of the effectiveness of the common requirements and quality attributes in covering the I.DOT One and I.DOT Mini systems. Then a concrete architecture of the I.DOT One will be presented. Finally, the results of implementing an I.DOT One prototype are presented.

6.1 Requirements of I.DOT One and I.DOT Mini

All of the common requirements in Table 5.1 covered the requirements of both the I.DOT One and I.DOT Mini. The core priority common requirements were needed to specify the systems requirements. As for the additional priority requirements some were taken and some were neglected depending on the system. In addition, further details about the devices' requirements had to be specified. For example, a device protocol in I.DOT One is different from I.DOT Mini. I.DOT One is a liquid dispensing device with eight dispensing channels, on the other hand, I.DOT Mini only has one. Similar detailed requirements differences can be seen to how liquids, plates, and device parameters are specified on each device. This does not mean that the common requirements are inadequate, they instead facilitated achieving the requirements.

6.2 Qualities Attributes of I.DOT One and I.DOT Mini

Unfortunately, the I.DOT One and I.DOT Mini did not have quality attributes and scenarios documented. Thus, there was nothing to compare the common scenarios against. Not a lot of effort have been made here to verify the common scenarios in Table 5.2. Fortunately, the common scenarios and core qualities is what the stakeholders captured about liquid handling devices. It should relate to both I.DOT One and I.DOT Mini and be taken as a starting point to specify further details about the system.

6.3 Concrete Architecture of I.DOT One

Figure 6.1 shows the concrete software architecture that was achieved using the client server SRA in Figure 5.5. As shown in the figure, only the relevant components for I.DOT One were taken from the client server SRA. For example, the cloud adapters were removed as they are not needed. Moreover, additional detailed software components were given to the architecture. For example, the architecture specifies the hardware actors that the I.DOT One system must interact with and the XML file type, in the file provider, that the system must be able to read. Reaching the concrete architecture was relatively simple and quick because the SRA presented an architectural layout containing the most important software components.

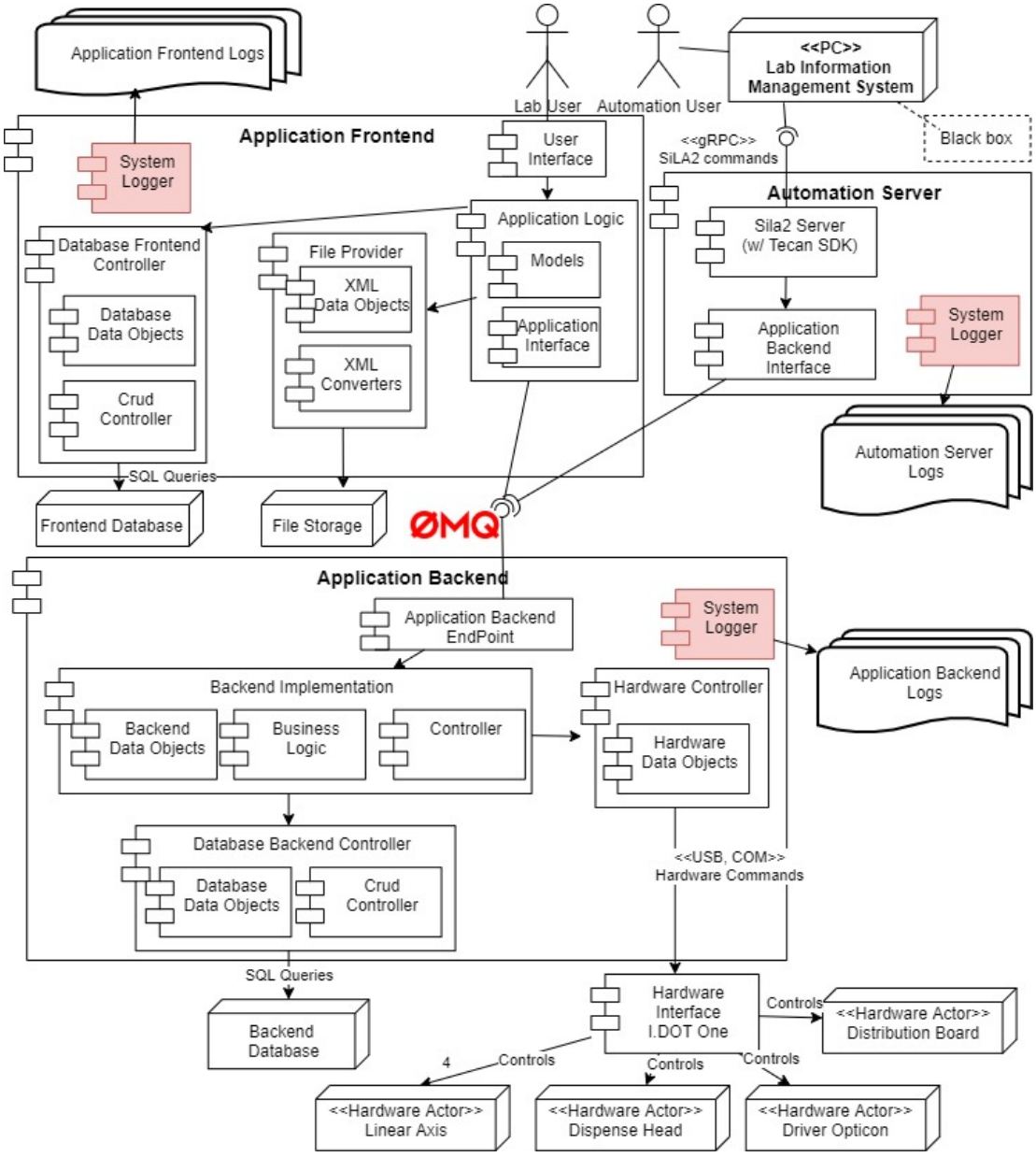

Figure 6.1: Software Architecture for IDOT.One

6.4 Prototype Implementation of I.DOT One

This section focuses on implementing a prototype using the I.DOT One concrete architecture in Figure 6.1. The prototype's main objective is to show that the SRA can be used to implement liquid handling devices, such as the I.DOT One. Due to time constraints, the prototype does not include the implementation of the components that are related to automation. Overall, using the architecture was simple to implement the prototype and it answers **RQ2**. The following sub-sections are comments about how different parts of the architecture were implemented.

6.4.1 Database

The database diagrams were used to create the database with SQL server. To implement data auditing (logging), SQL triggers were used to audit data in a separate database. The SQL triggers are convenient because they do not require any code to be written to handle audits. Moreover, triggers are implemented on the database layer which means that only one code call is required to save, update, and delete data as well as audit the data changes.

6.4.2 Application Backend

Implementing the backend was straight forward. No changes to the architecture were made. Each software component in the *Application Backend* had its own data objects. This is because the data objects in the database are represented differently than in the hardware or in the end-point. For example, a *bool* in the *end-point* is represented as a string in the database and can be represented as an *int* of 0 or 1 in the hardware components. The *end-point* also had a data transfer object which ZeroMQ will use to transfer data to the *Application Frontend*.

6.4.3 Communication Protocol/Framework

ZeroMQ, a messaging library, was used for the communication because it operates without a broker and offers request-reply and publish-subscribe messaging patterns. Both messaging patterns were needed because in same cases a client would like to request something from the server, and in other cases the liquid handling device needs to inform multiple clients about its status and results when executing a certain action. For example, the device might want to share the results of executing a device protocol in real time with multiple clients. Due to the lack of time, the performance scenarios were not extensively tested in the prototype. However, the prototype behaved in real time performance and ZeroMQ promises for asynchronous high-speed communication based on their documentation.

6.4.4 System logging

Since the prototype was implemented using .NET Core, SeriLog with Microsoft Logging were used to log system activities on files. Logging was implemented on a separate software component (or software project) to make sure that all other software components can reference it. Having a separate software component for logging is good since it decreases coupling and enabled the logging format to be changed by changing only one software component without affecting the others.

7 Conclusion and Future Work

An SRA is beneficial to reduce the amount of time and effort needed to achieve concrete software architecture and systems. However, there is not a specific evaluation method for achieving and evaluating SRAs. ATAM is a well-known method for architecture evaluation, but it has some disadvantages when applied towards a reference architecture. As a result, this thesis presented an adapted evaluation method for SRA that was derived from ATAM. The adapted evaluation method is focused towards achieving a reference architecture under a domain that will be identified by the stakeholders. The adapted evaluation method answered **RQ1** by providing a structure to follow to obtain a suitable SRA that can fulfill the domain and qualities the stakeholders look for. The adapted evaluation method includes how to define a domain, identify stakeholders, capture common requirements, and identify common quality attributes and scenarios. The method also provided an analysis method to compare different architectural approaches and SRAs by providing a score on how much they fulfill the requirements and scenarios.

The adapted evaluation method was proven to be beneficial when it was applied to the liquid handling device domain. The method helped stakeholders, architects, and software team, reach an SRA that is suitable for liquid handling devices. The output of the evaluation method includes tables that contain the common requirements, qualities, and scenarios that the stakeholders identified. It also includes the score of different architectural approaches to help chose which architectural approaches should be taken when designing the SRAs. The architect and the software team designed two potential SRAs: a client server and a message bus architecture. After scoring both of them, the client server SRA was followed. The SRA included how the device can be used locally and remotely, automated, and integrated with the cloud. The presented client server SRA answered **RQ2** because it captured the expected qualities and requirements that were provided by the stakeholders. For future work, it is better to apply the adapted evaluation method to several different domains to further verify its beneficial results as what this thesis proposed. Furthermore, the evaluation method should be applied by different software teams and architects to eliminate the possibility that the adapted evaluation method was beneficial for just a certain software team or architects.

The client server SRA will act as a facilitator to guide the software team and architects in achieving concrete software architecture and systems that are within the liquid handling device domain. To show the SRA's usage, I.DOT One and I.DOT Mini liquid handling devices were used as a case study. The requirements and qualities of the devices were specified based on the evaluation method's output. Moreover, a concrete architecture for I.DOT One was derived from the client server SRA. The I.DOT One architecture was then used to implement a small prototype to validate the architecture. Overall, the client server SRA facilitated the procedure in implementing the prototype. For future work, the SRA needs to be applied to more different concrete systems to better understand the flexibility of the SRA and its effectiveness in facilitating the concrete process to reach a liquid handling device system.

Bibliography

[20] "Supporting standards: SOPHIA KTORI DISCUSSES THE IMPLEMENTATION OF DATA STANDARDS FOR LABORATORY INFORMATICS." In: 170 (2020). URL: https://link.gale.com/apps/doc/A621580904/AONE?u=anon~ac77588a&sid=googleScholar&xid=a9a43e4a (cit. on p. 11).

[AAC+17] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster, P. Avgeriou. "A mapping study on design-time quality attributes and metrics". In: *Journal of Systems and Software* 127 (2017), pp. 52–77 (cit. on p. 24).

[AGG12] S. Angelov, P. Grefen, D. Greefhorst. "A framework for analysis and design of software reference architectures". In: (2012). DOI: 10.1016/j.infsof.2011.11.009 (cit. on pp. 14–16, 21).

[ATG08] S. Angelov, J. J. Trienekens, P. Grefen. "Towards a method for the evaluation of reference architectures: Experiences from a case". In: *European Conference on Software Architecture*. Springer. 2008, pp. 225–240 (cit. on pp. 18, 19).

[ATG14] S. Angelov, J. J. Trienekens, P. Grefen. "Extending and Adapting the Architecture Tradeoff Analysis Method for the Evaluation of Software Reference Architectures". In: (2014) (cit. on pp. 18, 19, 24).

[Bis13] P. Bisták. "Advanced remote laboratory for control systems based on Matlab and .NET platform". In: *2013 IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. 2013, pp. 35–39. DOI: 10.1109/ICETA.2013.6674400 (cit. on p. 11).

[Bro18] S. Brown. *Technical leadership and the balance with agility: Software Architecture for Developers - Volume 1*. 2018 (cit. on pp. 8, 12–14).

[BRST05] K. Bergner, A. Rausch, M. Sihling, T. Ternité. *DoSAM – Domain-Specific Software Architecture Comparison Model*. Springer Berlin Heidelberg NewYork, 2005, pp. 4–20. ISBN: 3540290338 (cit. on pp. 19, 23, 26).

[BZJ] M. A. Babar, L. Zhu, R. Jeffery. "A Framework for Classifying and Comparing Software Architecture Evaluation Methods". In: (). DOI: 10.17485/ijst/2016/v9i30/96653 (cit. on p. 18).

[CBB+10] P. Clements, F. Backmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. *Documenting Software Architectures: Views and Beyond*. 2nd edition. Addison-Wesley Educational Publishers Inc, 2010. ISBN: 0321552687 (cit. on p. 12).

[CBS17] M. A. Chauhan, M. A. Babar, Q. Z. Sheng. "A reference architecture for provisioning of tools as a service: meta-model, ontologies and design elements". In: *Future Generation Computer Systems* 69 (2017), pp. 41–65 (cit. on pp. 15, 16).

[CMV+09] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole, M. Bone. "The Concept of Reference Architectures". In: (2009). DOI: 10.1002/sys.20129 (cit. on pp. 9, 14, 15, 21).

[Dol01] T. J. Dolan. "Architecture Assessment of Information-System Families : a practical perspective". In: (2001) (cit. on pp. 21–23).

[Fai12] G. Fairbanks. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall Brainerd, 2012. ISBN: 987-0-9846181-0-1 (cit. on pp. 12–14).

[FGB04] E. Folmer, J. van Gurp, J. Bosch. *Software Architecture Analysis of Usability*. Springer, 2004, pp. 38–58. ISBN: 9783540260974 (cit. on pp. 22, 23, 26, 27, 29).

[GVV05] B. Graaf, H. Van Dijk, A. Van Deursen. "Evaluating an embedded software reference architecture-industrial experience report". In: *Ninth European Conference on Software Maintenance and Reengineering*. IEEE. 2005, pp. 354–363 (cit. on p. 19).

[ISO11] ISO/IEC. "ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models". In: (2011) (cit. on pp. 24, 38).

[KGFČ15] M. Kalúz, J. García-Zubía, M. Fikar, Ľ. Čirka. "A Flexible and Configurable Architecture for Automatic Control Remote Laboratories". In: *IEEE Transactions on Learning Technologies* 8.3 (2015), pp. 299–310. DOI: 10.1109/TLT.2015.2389251 (cit. on p. 11).

[KKC00] R. Kazman, M. Klein, P. Celements. "ATAM: Method for Architecture Evaluation". In: (2000). URL: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf (cit. on pp. 13, 19–21, 23–26, 28, 29).

[KYZC12] F. Kong, L. Yuan, Y. F. Zheng, W. Chen. "Automatic Liquid Handling for Life Science: A Critical Review of the Current State of the Art". In: *Journal of Laboratory Automation* 17.3 (2012). PMID: 22357568, pp. 169–185. DOI: 10.1177/2211068211435302. eprint: https://doi.org/10.1177/2211068211435302. URL: https://doi.org/10.1177/2211068211435302 (cit. on p. 11).

[LBK97] C.-H. Lung, S. Bot, K. Kalaichelvan. "An Approach to Software Architecture Analysis for Evolution and Reusability". In: (1997). URL: https://resources.sei.cmu.edu/asset_files/WhitePaper/1997_019_001_29775.pdf (cit. on pp. 21, 23, 24).

[LBVB00] N. Lassing, P. Bengtsson, H. van Vliet, J. Bosch. "Analyzing Software Architectures for Modifiability". In: (2000). URL: https://www.researchgate.net/publication/30499164_Analyzing_Software_Architectures_for_Modifiability (cit. on pp. 21, 23, 25, 26).

[MAFA13] S. Martínez-Fernández, C. Ayala, X. Franch, D. Anmeller. "A Framework for Software Reference Architecture Analysis and Review". In: (2013). URL: https://upcommons.upc.edu/bitstream/handle/2117/24040/eselaw2013_submission_14%20%282%29.pdf?sequence=1&isAllowed=y (cit. on p. 15).

[MGM06] M. Mattsson, H. Grahn, F. Mårtensson. "Software architecture evaluation methods for performance, maintainability, testability, and portability". In: *Second International Conference on the Quality of Software Architectures*. Citeseer. 2006 (cit. on p. 18).

[MSA+15] S. Martínez-Fernández, P. S. M. dos Santos, C. P. Ayala, X. Franch, G. H. Travassos. "Aggregating Empirical Evidence about the Benefits and Drawbacks of Software Reference Architectures". In: (2015). DOI: 10.1109/ESEM.2015.7321184 (cit. on p. 15).

[PS15a] A. Patidar, U. Suman. “A Survey on Software Architecture Evaluation Methods”. In: (2015) (cit. on p. 17).

[PS15b] A. Patidar, U. Suman. “A survey on software architecture evaluation methods”. In: *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE. 2015, pp. 967–972 (cit. on p. 18).

[PSL+20] M. Porr, S. Schwarz, F. Lange, L. Niemeyer, T. Hentrop, D. Marquard, P. Lindner, T. Scheper, S. Beutel. “Bringing IoT to the Lab: SiLA2 and Open-Source-Powered Gateway Module for Integrating Legacy Devices into the Digital Laboratory”. In: *HardwareX* 8 (2020), e00118. ISSN: 2468-0672. DOI: https://doi.org/10.1016/j.ohx.2020.e00118. URL: https://www.sciencedirect.com/science/article/pii/S2468067220300274 (cit. on p. 11).

[RW12] N. Rozanski, E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. 2nd edition. Pearson Education, Inc., 2012. ISBN: 9780321718334 (cit. on pp. 12–14, 17).

[RZRK19] A. Raza, S. Zafar, S. U. Rehman, U. Khattak. “Software Architecture Evaluation Methods: A Comparative Study”. In: *International Journal of Computing and Communication Networks* 1.2 (2019), pp. 1–9 (cit. on p. 18).

[SA17] I. Schmid, J. Aschoff. “A scalable software framework for data integration in bioprocess development”. In: *Engineering in Life Sciences* 17.11 (2017), pp. 1159–1165. DOI: https://doi.org/10.1002/elsc.201600008. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/elsc.201600008. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/elsc.201600008 (cit. on p. 11).

[Sol12] F. Solms. *What is Software Architecture?* Association for Computing Machinery, 2012, pp. 363–373. ISBN: 9781450313087 (cit. on p. 12).

[SS12] P. Shanmugapriya, R. Suresh. “Software architecture evaluation methods-a survey”. In: *International Journal of Computer Applications* 49.16 (2012) (cit. on p. 18).

[ZSZ11] S. Zhao, Z. Shi, S. Zhu. “A virtual laboratory architecture for engineering education”. In: *2011 IEEE 3rd International Conference on Communication Software and Networks*. 2011, pp. 560–563. DOI: 10.1109/ICCSN.2011.6013895 (cit. on p. 11).

All links were last followed on September 27, 2021.

A Appendix

A.1 Applicable Evaluation Methods

Table A.1 contains the evaluation methods that are applicable to be studied when designing an evaluation method for SRA. It also contains the reasons behind the method's applicability and its usability for the SRA evaluation method.

Table A.1: Applicable Evaluation Methods

Applicable Evaluation Method	**Reasons for Applicability**	**Usability for the SRA evaluation method**
ATAM	Well documented and used method for architecture design tradeoffs. Evaluates and captures multiple quality attributes and scenarios. Provides risks, sensitivity points, and tradeoffs for architecture decision.	As a base foundation for the SRA evaluation method.
ALMA	Evaluates the modifiability quality attribute and obtains change scenarios. The evaluation method goal can be to compare different architectures.	Refine the evaluation steps of obtaining modifiability quality attributes and change scenarios, and to take ideas on how to do architecture comparison.
DoSAM	Evaluates multiple quality attributes. The evaluation method goal is to compare different architectures.	Take ideas on how to do architecture comparison.
FAAM	Focuses on interoperability and extensibility quality attribute.	Ideas on how to capture and evaluate extensibility quality attribute are taken.
SAAM for Complex Scenarios (SAAMCS)	Focuses on flexibility quality attribute. Evaluated based on scenarios related to flexibility. Applied to the final architecture version.	Ideas on how to capture and evaluate flexibility quality attribute are taken.
SALUTA	An addition to ATAM that helps in obtaining and evaluating the usability quality attributes and scenarios.	Ideas on how to capture and evaluate usability quality attribute are taken.
SAAMER	Method to identify if enough scenarios are available for each quality attribute.	Check whether enough scenarios have been captured.

Continued on next page

Table A.1 – continued from previous page

Applicable Evaluation Method	Reasons for Applicability	Usability for the SRA evaluation method
ISO 42030	A standard that contains the things that need to be considered when having an evaluation method.	Take ideas of what the evaluation method should contain.
Architecture Tradeoff Analysis Method for Reference architectures (ATAM\R)	An extended ATAM approach to evaluate SRA for multiple organizations. Designed to evaluate SRA designed for multiple organizations.	Take ideas on how it extends ATAM, to find better approaches and improvements to evaluate SRA for a single organization.

A.2 In-Applicable Evaluation Methods

Table A.2 contains the evaluation methods that are not suitable as a base or a reference to design an evaluation method for SRA. It also contains the reasons behind the decision.

Table A.2: Inapplicable Evaluation Methods

Inapplicable Evaluation Method	Reasons for In-applicability
SAAM	ATAM is an improvement on SAAM.
Extending SAAM by Integration in the Domain (ESAAMI)	The method focuses on reusing assets and activities which is not applicable for SRA. The method is also still not mature.
Architecture Level Prediction of Software Maintenance (ALPSM)	Evaluation based on previous maintenance data (such as cost and time) which is not possible to obtain for this paper's case.
Scenario-Based Architecture Re-engineering (SBAR)	Short iterative approach to re-engineer an existing developed system.
Cost Benefit Analysis Method (CBAM)	Focuses on expenses and profits quality attributes which is not the focus of the SRA evaluation method.
Software Architecture Comparison Analysis Method (SACAM)	DoSAM is an improvement on SACAM.
RARE	Tends toward evaluating the usability of the code.
Empirically-Based Architecture Evaluation (EBAE)	Compares architectures after a code change.
Software Performance Engineering (SPE)	Analyzes the performance of the system in regards to executing operations.
Aspectual Software Architecture Analysis Method (ASAAM)	Focuses on refactoring the architecture.
Creative Innovative Patterns for Architecture Analysis (CIPA)	Improves the quality attributes of existing software architecture by refining them.

Continued on next page

Table A.2 – continued from previous page

Inapplicable Evaluation Method	Reasons for Applicability
Active Reviews for Intermediate Design (ARID)	Fills a niche in the design review spectrum.
Integrating SAAM in Domain-Centric and Reuse-Based (ISAAMCR)	Designed heavily to evaluate concrete architecture and focuses on the re-usability aspect of a concrete architecture.

A.3 Architectural Approaches for Liquid Handling Devices

Table A.3 contains all the different architectural approaches that were considered for liquid handling devices according to a cluster of scenarios. The cluster name is shown at the top of the cluster of scenarios.

Table A.3: Architectural Approaches for Liquid Handling Devices

Scenario	Architectural Approach
SiLA2	
Device can be operated via SiLA2.	Set the SiLA2 commands using Tecan SDK.
Implement, test, and deploy a new SiLA2 endpoint automation command in < 3 person day.	Set the SiLA2 commands using the provided skeleton example from the SiLA community.
Implement, test, and deploy a new communication protocol endpoints to the system (For example: AutoIT) in < 3 person weeks.	Implement internally an SDK that sets SiLA2 commands.
Audit Logging	
Log all the changes, and by whom they were made, for any protocol at any point in time.	Log on files (File management system)
Log all the protocols that were executed with their results and device settings that were used.	Log on database
Log all changes and by whom they were made, to the device hardware and software settings.	
Log all login details (failed/successful attempts, when logged-in and logged-out).	
Device log size should not exceed 500 Mbs.	
Change log format structure in < 2 person days.	
Data	
Adapt a new protocol format in < 1 person week (Ex: adding JSON as a new format).	Save data on a file-based system.
Implement (or modify), test, and deploy a protocol format structure in < 1 person week (Ex: changing plate names, introducing a new plate type).	Save on SQL DB.

Continued on next page

Table A.3 – continued from previous page

Scenario	Architectural Approach
Opening (loading) a protocol locally should not exceed 15 seconds.	Save on NoSQL DB.
The protocol is only accessible to the authorized group specified by the protocol group.	
The protocol can be hashed with an electronic signature which is saved on both the protocol and the DB, and it cannot be copied.	
All data (database information) including log files can be encrypted in production.	
Backup entire database in <1 person day.	
Protocol, liquid, and plate data can be archived and unarchived in less than 15 seconds.	
The system should be able to run protocols, config files, and all data related files that are at least 2 versions behind.	
Only users that have access to a protocol, can view the changes that were made to the protocol and when was the protocol executed with the execution results.	
Implement, test, and deploy a new parameter to the device settings in < 3 person days.	
Communication to hardware	
Implement (or modify), test, and deploy a software end point and logic to a hardware component in <1 person week.	Implement an individual interface/gateway for each hardware component.
	Implement one interface/gateway that communicates with all the hardware components.
	Implement a software bus for all the hardware components.
Communication Protocol / Framework	
When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	REST
The system must receive and send data in realtime over a network.	gRPC
The remote system (that is connected to the device) must be independent from the framework used to build it.	MQTT

Continued on next page

Table A.3 – continued from previous page

Scenario	Architectural Approach
The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	Apache Kafka
An unauthorized person should not be able to login to the device and use it.	RabbitMQ
Only user accounts with access to automation are allowed to run the automation commands.	ZeroMQ
	Apache ActiveMQ
Firmware	
Update device firmware should take less than 3 minutes.	Device firmware is saved on the device storage.
	Device firmware is saved on an external DB.
UI	
Integrate a completely new software UI design in < 10 weeks of work (Ex: Only desktop UI apps are available, and we would like to integrate a new website UI to the system).	Split the (local and remote) UI from the rest of the logic using an API.
	Have the local UI be binded to the logic and the remote UI binded to objects that get their data from an API.

A.4 Architectural Approaches Comparison for Liquid Handling Devices

Table A.4 contains the advantages and disadvantages of each architectural approach for liquid handling devices. The cluster name is shown at the top of the cluster of scenarios.

Table A.4: Architectural Approaches Advantages and Disadvantages for Liquid Handling Devices

Architectural Approach	Advantages	Disadvantages
SiLA2		
Set the SiLA2 commands using Tecan SDK.	Tecan SDK is easier to initially set commands (maintainability). Tecan SDK auto generates the needed XML files and classes for SiLA. The Tecan SDK license (BSD 3-Clause) allows to modify it in private use.	Dependent on a 3rd party license (maintainability - Legality).

Continued on next page

Table A.4 – continued from previous page

Architectural Approach	Advantages	Disadvantages
Set the SiLA2 commands using the provided skeleton example from the SiLA community.		More time consuming than using Tecan SDK.
Implement internally an SDK that sets SiLA2 commands.	No licensing issues because SDK is implemented internally.	Will take longer to implement compared to the other approaches.
Audit Logging		
Log on files (File management system)	The audit log can be easily shared and copied if needed.	Hard to keep track of the log files for multiple device protocols. File has to be locked to ensure that the user cannot change the log manually. A change in the log format structure will result in log files having different format from the previous versions.
Log on database	Can have a structured database that contains the audit log of the device protocols. Log format depends on how the audit log is retrieved from the database.	Require the device protocols to be saved on the database and not in the file storage. Additional effort is required to retrieve the audit from the database.
Data		
Save data on a file based system.	Can encrypt files if needed.	No structure. Cannot easily archive data. Cannot easily audit data. Cannot easily apply permissions to data.
Save on SQL DB.	Can encrypt data if needed. Provide a structure for the data. Can easily archive to another DB. Can easily apply permissions to data. Can easily implement data audit using SQL triggers.	Need a translator that will translate file protocols to be saved in the DB.
Save on NoSQL DB.	Can encrypt data if needed. Can easily archive to another DB. Can easily implement data audit using NoSQL triggers. Can easily apply permissions to data.	No structure. Code logic has to be implemented.

Continued on next page

Table A.4 – continued from previous page

Architectural Approach	Advantages	Disadvantages
Communication to hardware		
Implement an individual interface/gateway for each hardware component.	Increases loose-coupling since each hardware component has its own interface.	The software must reference each interface.
Implement one interface/gateway that communicates with all the hardware components.	The software will receive inputs from one location (A single reference for the software to the interface).	Increases coupling.
Implement a software bus for all the hardware components.	The software will receive inputs from one location (A single reference for the software to the bus).	Troubleshooting of individual device is difficult. The bus should handle the required load from all the components.
Communication Protocol / Framework		
REST	Offers request reply messaging pattern.	Multiple clients listening to the same message have to be implemented manually.
gRPC	Offers request reply, client streaming, server streaming, and bidirectional streaming messaging patterns.	Does not allow multiple clients to listen to the same message.
MQTT	Offers request reply, publish subscribe, broadcast, and fan-in messaging patterns. Message broker offers logging.	Need to deploy an additional component which is the message broker. Additional effort is needed to set the authentication between the message broker and the clients.
Apache Kafka	Offers publish subscribe messaging pattern. Message broker offers logging.	Need to deploy an additional component which is the message broker. Additional effort is needed to set the authentication between the message broker and the clients.
RabbitMQ	Offers publish subscribe messaging pattern. Message broker offers logging. Supports topic authorization which can help implement an RBAC solution.	Need to deploy an additional component which is the message broker. Additional effort is needed to set the authentication between the message broker and the clients.
ZeroMQ	Offers request reply, publish subscribe, push pull (pipeline), and pair messaging patterns.	

Continued on next page

Table A.4 – continued from previous page

Architectural Approach	Advantages	Disadvantages
Apache ActiveMQ		Offers a message-oriented middleware (MOM) which is not needed.
Firmware		
Device firmware is saved on the device storage.	Device can run on its own and get the firmware without the need of any external connections to get the firmware.	
Device firmware is saved on an external DB.		Device always needs a connection to the DB to get the firmware (may affect performance).
UI		
Split the (local and remote) UI from the rest of the logic using an API.	Decreases coupling. Easier to maintain and debug in the long run.	Takes more time to develop.
Have the local UI be binded to the logic and the remote UI binded to objects that get their data from an API.	Easier and faster to develop.	Increases coupling. Harder to maintain in the long run.

A.5 Architectural Approaches Weighted Score for Liquid Handling Devices

Table A.5 contains the weighted score of each architectural approach within the scenario cluster. The chosen architectural approaches are the ones with the highest total weighted score and are marked in red color.

Table A.5: Architectural Approaches Weighted Score

Architectural Approach	Scenario	Priority	Score	Weighted Score
SiLA2				
Set the SiLA2 commands using Tecan SDK.	Device can be operated via SiLA2.	H	5	15
	Implement, test, and deploy a new SiLA2 endpoint automation command in < 3 person day.	H	5	15
	Implement, test, and deploy a new communication protocol endpoints to the system (For example: AutoIT) in < 3 person weeks.	L	5	5

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
			Total	**35**
Set the SiLA2 commands using the provided skeleton example from the SiLA community.	Device can be operated via SiLA2.	H	5	15
	Implement, test, and deploy a new SiLA2 endpoint automation command in < 3 person day.	H	4	12
	Implement, test, and deploy a new communication protocol endpoints to the system (For example: AutoIT) in < 3 person weeks.	L	5	5
			Total	**32**
Implement internally an SDK that sets SiLA2 commands	Device can be operated via SiLA2.	H	5	15
	Implement, test, and deploy a new SiLA2 endpoint automation command in < 3 person day.	H	3	9
	Implement, test, and deploy a new communication protocol endpoints to the system (For example: AutoIT) in < 3 person weeks.	L	5	5
			Total	**29**
Audit Logging				
Log on files (File management system)	Log all the changes, and by whom they were made, for any protocol at any point in time.	H	1	3
	Log all the protocols that were executed with their results and device settings that were used.	H	3	9
	Log all changes and by whom they were made, to the device hardware and software settings.	H	3	9
	Logs all login details (failed/successful attempts, when logged-in and logged-out).	H	3	9
	Device log size should not exceed 500 Mbs.	L	5	5
	Change log format structure in < 2 person days.	L	3	3

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
			Total	**38**
Log on database	Log all the changes, and by whom they were made, for any protocol at any point in time.	H	4	12
	Log all the protocols that were executed with their results and device settings that were used.	H	5	15
	Log all changes and by whom they were made, to the device hardware and software settings.	H	5	15
	Logs all login details (failed/successful attempts, when logged-in and logged-out).	H	5	15
	Device log size should not exceed 500 Mbs.	L	5	5
	Change log format structure in < 2 person days.	L	5	5
			Total	**67**
Data				
Save data on a file based system	Adapt a new protocol format in < 1 person week (Ex: adding JSON as a new format).	M	5	10
	Implement (or modify), test, and deploy a protocol format structure in < 1 person week (Ex: changing plate names, introducing a new plate type).	H	2	6
	Opening (loading) a protocol locally should not exceed 15 seconds.	H	5	15
	The protocol is only accessible to the authorized group specified by the protocol group.	H	2	6
	The protocol can be hashed with an electronic signature which is saved on both the protocol and the DB, and it cannot be copied.	H	3	9
	All data (database information) including log files can be encrypted using XX in production.	H	4	12

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
	Backup entire database in <1 person day.	L	4	4
	Protocol, liquid, and plate data can be archived and unarchived in less than 15 seconds.	H	4	12
	The system should be able to run protocols, config files, and all data related files that are at least 2 versions behind.	M	5	10
	Only users that have access to a protocol, can view the changes that were made to the protocol and when was the protocol executed with the execution results.	M	2	4
	Implement, test, and deploy a new parameter to the device settings in < 3 person days.	M	5	10
			Total	**98**
Save on SQL DB	Adapt a new protocol format in < 1 person week (Ex: adding JSON as a new format).	M	2	4
	Implement (or modify), test, and deploy a protocol format structure in < 1 person week (Ex: changing plate names, introducing a new plate type).	H	5	15
	Opening (loading) a protocol locally should not exceed 15 seconds.	H	5	15
	The protocol is only accessible to the authorized group specified by the protocol group.	H	5	15
	The protocol can be hashed with an electronic signature which is saved on both the protocol and the DB, and it cannot be copied.	H	4	12
	All data (database information) including log files can be encrypted using XX in production.	H	5	15
	Backup entire database in <1 person day.	L	5	5

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
	Protocol, liquid, and plate data can be archived and unarchived in less than 15 seconds.	H	5	15
	The system should be able to run protocols, config files, and all data related files that are at least 2 versions behind.	M	5	10
	Only users that have access to a protocol, can view the changes that were made to the protocol and when was the protocol executed with the execution results.	M	5	10
	Implement, test, and deploy a new parameter to the device settings in < 3 person days.	M	5	10
			Total	**126**
Save on NoSQL DB	Adapt a new protocol format in < 1 person week (Ex: adding JSON as a new format).	M	2	4
	Implement (or modify), test, and deploy a protocol format structure in < 1 person week (Ex: changing plate names, introducing a new plate type).	H	5	15
	Opening (loading) a protocol locally should not exceed 15 seconds.	H	5	15
	The protocol is only accessible to the authorized group specified by the protocol group.	H	4	12
	The protocol can be hashed with an electronic signature which is saved on both the protocol and the DB, and it cannot be copied.	H	4	12
	All data (database information) including log files can be encrypted using XX in production.	H	5	15
	Backup entire database in <1 person day.	L	5	5
	Protocol, liquid, and plate data can be archived and unarchived in less than 15 seconds.	H	5	15

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
	The system should be able to run protocols, config files, and all data related files that are at least 2 versions behind.	M	5	10
	Only users that have access to a protocol, can view the changes that were made to the protocol and when was the protocol executed with the execution results.	M	4	8
	Implement, test, and deploy a new parameter to the device settings in < 3 person days.	M	5	10
			Total	**121**
Communication to hardware				
Implement an individual interface/gateway for each hardware component.	Implement (or modify), test, and deploy a software end point and logic to a hardware component in <1 person week.	H	4	12
			Total	**12**
Implement one interface/-gateway that communicates with all the hardware components.	Implement (or modify), test, and deploy a software end point and logic to a hardware component in <1 person week.	H	2	6
			Total	**6**
Implement a software bus for all the hardware components.	Implement (or modify), test, and deploy a software end point and logic to a hardware component in <1 person week.	H	5	15
			Total	**15**
Communication Protocol / Framework				
REST	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H	5	15
	The system must receive and send data in real time over a network.	H	5	15
	The remote system (that is connected to the device) must be independent from the framework used to build it.	M	4	8

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H	3	9
	An unauthorized person should not be able to login to the device and use it.	H	2	6
	Only user accounts with access to automation are allowed to run the automation commands.	H	1	3
			Total	**56**
gRPC	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H	5	15
	The system must receive and send data in real time over a network.	H	5	15
	The remote system (that is connected to the device) must be independent from the framework used to build it.	M	4	8
	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H	2	6
	An unauthorized person should not be able to login to the device and use it.	H	5	15
	Only user accounts with access to automation are allowed to run the automation commands.	H	1	3
			Total	**62**
MQTT	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H	5	15
	The system must receive and send data in real time over a network.	H	5	15

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
	The remote system (that is connected to the device) must be independent from the framework used to build it.	M	3	6
	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H	4	12
	An unauthorized person should not be able to login to the device and use it.	H	5	15
	Only user accounts with access to automation are allowed to run the automation commands.	H	1	3
			Total	**66**
Apache Kafka	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H	5	15
	The system must receive and send data in real time over a network.	H	5	15
	The remote system (that is connected to the device) must be independent from the framework used to build it.	M	3	6
	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H	4	12
	An unauthorized person should not be able to login to the device and use it.	H	5	15
	Only user accounts with access to automation are allowed to run the automation commands.	H	2	6
			Total	**69**

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
RabbitMQ	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H	5	15
	The system must receive and send data in real time over a network.	H	5	15
	The remote system (that is connected to the device) must be independent from the framework used to build it.	M	3	6
	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H	4	12
	An unauthorized person should not be able to login to the device and use it.	H	5	15
	Only user accounts with access to automation are allowed to run the automation commands.	H	2	6
			Total	**69**
ZeroMQ	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H	5	15
	The system must receive and send data in real time over a network.	H	5	15
	The remote system (that is connected to the device) must be independent from the framework used to build it.	M	4	8
	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H	5	15
	An unauthorized person should not be able to login to the device and use it.	H	5	15

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
	Only user accounts with access to automation are allowed to run the automation commands.	H	1	3
			Total	**71**
Apache ActiveMQ	When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time.	H	5	15
	The system must receive and send data in real time over a network.	H	5	15
	The remote system (that is connected to the device) must be independent from the framework used to build it.	M	4	8
	The hardware system should allow more than one client system to subscribe to it. More than one client can receive the hardware reply messages.	H	1	3
	An unauthorized person should not be able to login to the device and use it.	H	5	15
	Only user accounts with access to automation are allowed to run the automation commands.	H	1	3
			Total	**59**
Firmware				
Device firmware is saved on the device storage.	Update device firmware should take less than 3 minutes.	L	5	5
			Total	**5**
Device firmware is saved on an external DB.	Update device firmware should take less than 3 minutes.	L	1	1
			Total	**1**
UI				
Split the (local and remote) UI from the rest of the logic using an API.	Integrate a completely new software UI design in < 10 weeks of work (Ex: Only desktop UI apps are available, and we would like to integrate a new website UI to the system).	L	5	5

Continued on next page

Table A.5 – continued from previous page

Architectural Approach	Scenario	Priority	Score	Weighted Score
			Total	**5**
Have the local UI be binded to the logic and the remote UI binded to objects that get their data from an API.	Integrate a completely new software UI design in < 10 weeks of work (Ex: Only desktop UI apps are available, and we would like to integrate a new website UI to the system).	L	1	1
			Total	**1**

A.6 Architectural Design Comparison

Table A.6 only shows the scenarios that had a different score between the client server architecture in Figure 5.5 and the message bus architecture in Figure 5.6. The client server architecture has a higher total weighted score than the message bus architecture.

Table A.6: Architectural Design Weighted Score

Scenario	Priority	Quality Attribute	Architectural Design	Score	Weighted Score
Device can be operated via SiLA2	H	Compatibility - Interoperability	Client Server	5	15
Adapt a new protocol format in < 1 person week (Ex: adding JSON as a new format)	M	Maintainability - Modifiability	Client Server	3	6
Implement (or modify), test, and deploy a protocol format structure in < 1 person week (Ex: changing plate names, introducing a new plate type)	H	Maintainability - Modifiability	Client Server	3	9
Implement (or modify), test, and deploy a software end point and logic to a hardware component in <1 person week	H	Maintainability - Modifiability	Client Server	4	12
When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time	H	Performance - Time behaviour	Client Server	5	15
The system must receive and send data in real time over a network	H	Performance - Time behaviour	Client Server	5	15

Continued on next page

Table A.6 – continued from previous page

Scenario	Priority	Quality Attribute	Architectural Design	Score	Weighted Score
Logs all login details (failed/successful attempts, when logged-in and logged-out)	H	Security - Accountability	Client Server	5	15
				Total	**87**
Device can be operated via SiLA2	H	Compatibility - Interoperability	Message Bus	2	6
Adapt a new protocol format in < 1 person week (Ex: adding JSON as a new format)	M	Maintainability - Modifiability	Message Bus	4	8
Implement (or modify), test, and deploy a protocol format structure in < 1 person week (Ex: changing plate names, introducing a new plate type)	H	Maintainability - Modifiability	Message Bus	5	15
Implement (or modify), test, and deploy a software end point and logic to a hardware component in <1 person week	H	Maintainability - Modifiability	Message Bus	3	9
When accessed remotely, the UI shall be able to interact with the device in real-time without having visible delays for 95% of the interaction time	H	Performance - Time behaviour	Message Bus	4	12
The system must receive and send data in real time over a network	H	Performance - Time behaviour	Message Bus	4	12
Logs all login details (failed/successful attempts, when logged-in and logged-out)	H	Security - Accountability	Message Bus	3	9
				Total	**71**

A.7 Frontend Database Diagram

Figure A.1 is the database diagram that will be deployed on the laboratory user's PC. This database is going to be only accessible to the respective user. Please note that the diagram does not include all the table columns. It is a reference database diagram to facilitate the design of the concrete database diagram.

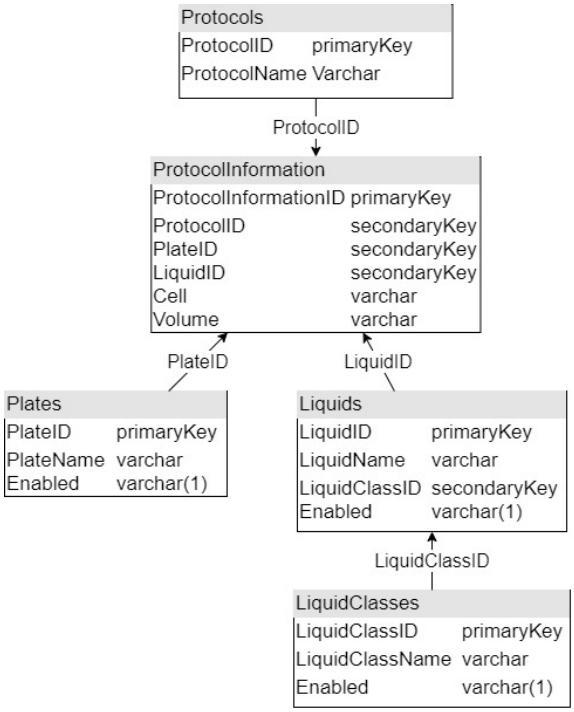

Figure A.1: Frontend Database Diagram

A.8 Backend Database Diagram

Figure A.2 is the database that will be deployed on the liquid handling device. Please note that the diagram does not include all the table columns. It is a reference database diagram to facilitate the design of the concrete database diagram. Also, if the database is going to be deployed on a centralized machine, then only the hardware settings need to be deployed on the liquid handling device. The rest will be deployed on the centralized machine.

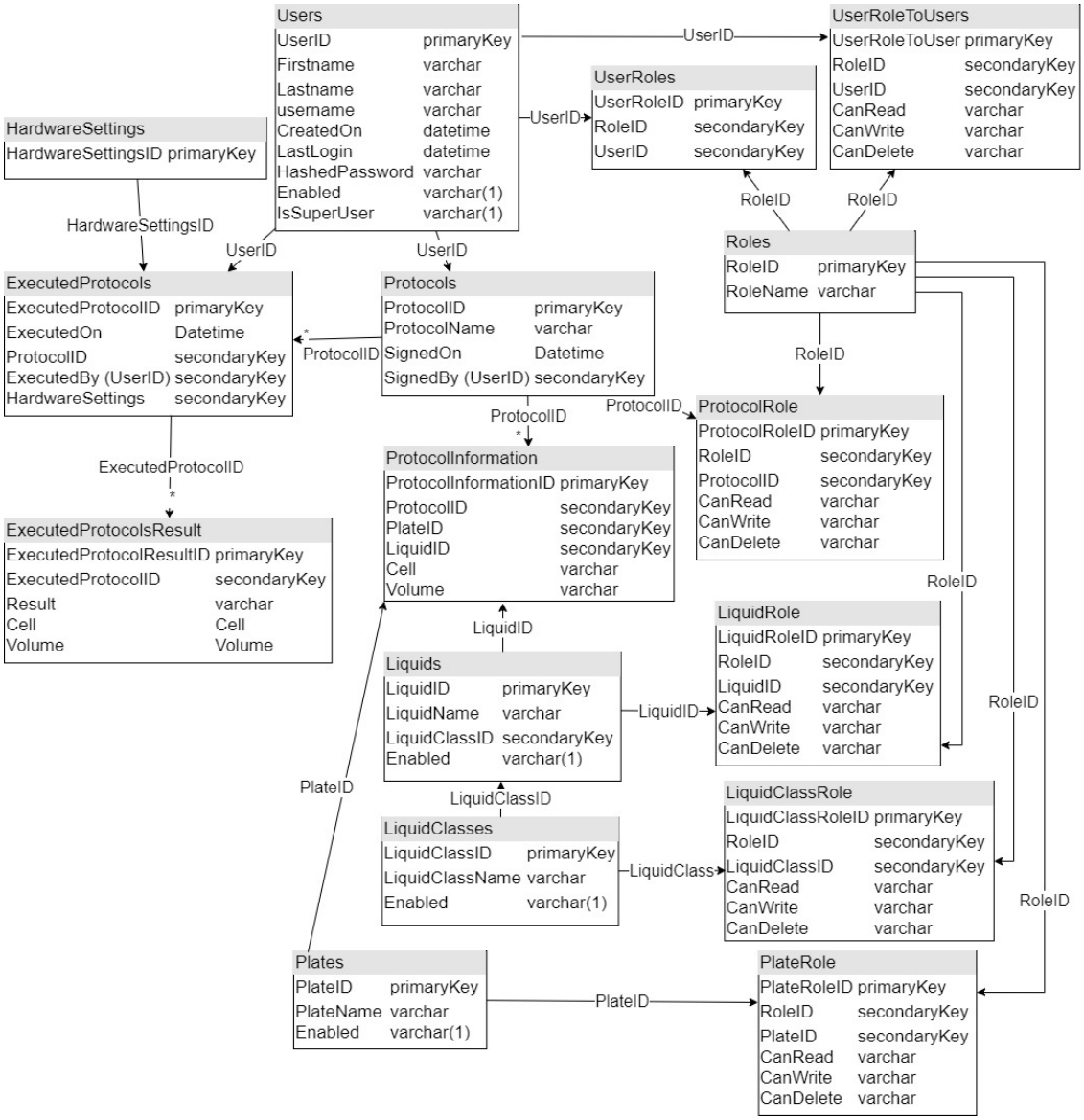

Figure A.2: Backend Database Diagram

A.9 Audit Database Diagram

Figure A.3 is the audit database that will be deployed on the liquid handling device to audit all the data changes that are done to the backend database diagram. Auditing will be done using SQL triggers to insert data in the audit database. The audit database is similar to the backend database except it has additional columns on each table to indicate whom did the data change, when was the data change made, and what was the data change. Please note that the diagram does not include all the table columns.

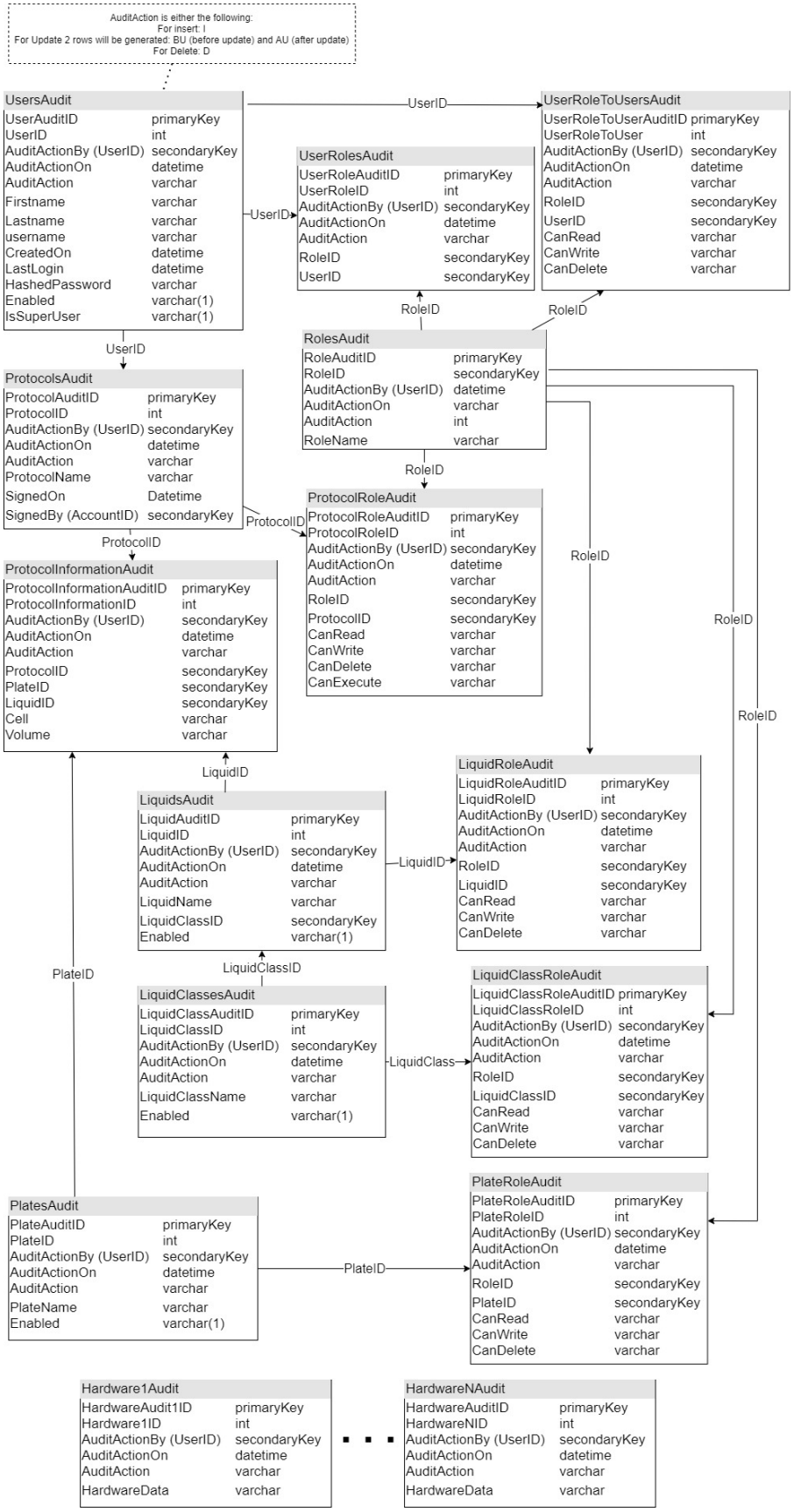

Figure A.3: Audit Database Diagram