MASTER

Sound camera as a service

van Vliet, T.

Award date:
2016

Link to publication
Sound camera as a service
Master thesis

Tijl van Vliet
t.v.vliet@student.tue.nl

February 2, 2016

Sorama B.V.
Eindhoven

Publishing date: 29 January 2017

System Architecture and Networking
Department of Mathematics and Computer Science
Eindhoven University of Technology

Assessment committee:
- dr. Rudolf Mak (TU/e supervisor)
- ir. Wouter Ouwens (Company supervisor)
- dr. Kees Huizing
Abstract

The World Health Organization (WHO) concluded that noise pollution is linked to a range of health problems. The biggest problem is the loss of sleep, the WHO advises that background noise should be on average 30 decibel and should not exceed 45 decibel for a good night’s sleep. In Europe around 70 million people are exposed to daily vibrations and noise levels of 55 decibels or more from traffic alone.

The research described in this report is held within the company Sorama who is specialized in making noise and vibration visible. For this purpose they have designed a Sound Camera which contains 1024 MEMS microphones. The development of this Sound Camera is an ongoing improvement. Currently, a version with new more powerful hardware is developed that can support more applications. To achieve the latter Sorama needs to create a new flexible software architecture. In the context of this project the usage of the Sound Camera is restricted to product development by Sorama’s customers. Sorama is always looking for new applications in different market segments whereby the Sound Camera can be used, for example in smart cities. Therefor the architecture needs to be flexible and extensible with future functionalities.

This research classifies the Sound Camera as an iot device, due to its ability to connect to the Sorama Portal, the Sorama server environment which is responsible for the computation and analysis of the data acquired by the Sound Camera. The design of the new software architecture is based on a Service Oriented Architecture (SOA) pattern. SOA allows the software architecture to define all the different possible functionalities of the Sound Camera as a combination of services. This means new functionalities can be added by creating new services, or composing existing services in a different manner. The design allows services to be changed on-the-fly, making it an optimal flexible software architecture for the Sorama Sound Camera.

To demonstrate the workings and practicability of the new software architecture, a proof of concept implementation is made. This proof of concept is deployed on the new hardware design of the Sound Camera. It allows the users to control and operate the Sound Camera from a web-browser and record and analyze measurements.
Acknowledgement

In front of you you find the crown jewel after almost 5 years studying at the TU/e. It is the closure and the start of a new beginning, what is also called: ‘Normal life’ instead of staying ‘the eternal student’. During my years at the TU/e I learned more than I ever could image, not only knowledge, but also what I want to become in my further live.

I am very grateful for the opportunity Sorama gave me to do my Master Thesis with them. I don’t know if I would have finished my study, if I had done it internally at the TU/e. I would like to thank the entire team of Sorama of giving me a memory I will never forget. Especially Wouter, I want to thank you for your angelic patience and all the guidance to achieve this. The team of Sorama, they don’t feel as co-workers but as friends.

I want to thank dr. Rudolf for all the help, advise and support during the project. I know I was not the standard track student at the TU/e therefor needed some hard rules and sometimes bringing me back to the hard reality. If you would not have done that, I think I would not have ever finished it. I would also want to thank all the other employees of the TU/e that helped me trough all the paper work and the bureaucracy.

In particular, I would really like to thank Floris for all the patience and crap he had to go through. I know it was already a hard discussion on its own to leave behind your security of a job and start a new adventure of our own Company together. Things did not go as fast as we expected, because I wasn’t able to finish my thesis last summer, but within a few weeks our own Company is a fact. So again thank you Floris.

I want to thank in alphabetic order Babette, Kenny and Sem for their guidance and help with my report.
Finally, I would like to thank all my family and friends, because without their support and patience I would never have done it. But it now finally happens.
# Contents

Abstract ii

Acknowledgement iv

Acronyms x

List of Figures xii

List of Tables xiv

Listings xvi

1 Introduction 1

1.1 Context/Company/Application domain/Scenario 1

1.1.1 Context 1

1.1.2 Company 1

1.1.3 Application domain 1

1.1.4 Scenario 1

1.2 Algorithms 3

1.2.1 Beamforming 3

1.2.2 Near-field Acoustic Holography 4

1.2.3 Similarities of the techniques 4

1.3 Software 4

1.3.1 Current Software Architecture 4

1.3.2 Data generated 6

1.4 Hardware 6

1.4.1 Current Design 7

1.4.2 New Design 7

1.4.3 External trigger 8

1.5 Problem statement 10

1.6 Goals 10

1.7 Outline/Overview 10

2 Related Work 11

2.1 Architecture patterns 13

2.1.1 Component Based Architecture 13

2.1.2 Event-driven Architecture 13

2.1.3 Service Oriented Architecture 13

3 Requirements Analysis 15

3.1 Analysis 15

3.2 Perspectives 15

3.2.1 User perspective 15

3.2.2 Company perspective 16

3.2.3 Technical perspective 17

3.3 Requirements 17

3.3.1 User 17

3.3.2 Software 18
4 Architecture and design

4.1 Software System Architecture

4.2 Architecture pattern

4.3 Selection of default components and services

4.4 Stakeholders

4.5 Viewpoints

4.5.1 Context viewpoint

4.5.2 Functional viewpoint

4.5.3 Development viewpoint

4.5.4 Deployment viewpoint

4.6 Perspectives

4.6.1 Availability and Resilience perspective

4.6.2 Development Resource perspective

4.6.3 Usability perspective

4.6.4 Security perspective

5 Implementation

5.1 Selection of implementation

5.1.1 Programming Language

5.1.2 Discovery Service

5.1.3 Communication Service

5.1.4 ServiceBus

5.1.5 Wire protocol

5.1.6 Software as a Service

5.2 Design of proof of concept

5.2.1 Design principles

5.2.2 Class diagram

5.2.3 Listing of interfaces

5.2.4 Code snippet

5.2.5 Sequence diagram

5.3 Use cases

5.3.1 Connecting multiple ServiceBusses

5.3.2 Doing Measurements

5.3.3 Synchronizing Clock

5.3.4 Making offline measurements

5.3.5 Deviating communication

5.4 Development board

6 Validation & Evaluation

6.1 Experiments

6.1.1 Experiment Discovery

6.1.2 Experiment Measurement

6.1.3 Experiment Analyze

6.1.4 Experiment Repeatability

6.2 Validation of Requirements

6.2.1 User Requirements

6.2.2 Software Requirements
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.3</td>
<td>49</td>
</tr>
<tr>
<td>6.3</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>53</td>
</tr>
<tr>
<td>7.1</td>
<td>53</td>
</tr>
<tr>
<td>7.2</td>
<td>54</td>
</tr>
<tr>
<td>References</td>
<td>55</td>
</tr>
<tr>
<td>Appendix A</td>
<td>59</td>
</tr>
<tr>
<td>Appendix B</td>
<td>63</td>
</tr>
<tr>
<td>Appendix C</td>
<td>65</td>
</tr>
<tr>
<td>C.1</td>
<td>65</td>
</tr>
<tr>
<td>C.2</td>
<td>66</td>
</tr>
<tr>
<td>C.3</td>
<td>67</td>
</tr>
</tbody>
</table>
### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>Architecture Description.</td>
</tr>
<tr>
<td>ARM</td>
<td>Acorn RISC Machine.</td>
</tr>
<tr>
<td>CLR</td>
<td>Common Language Runtime.</td>
</tr>
<tr>
<td>DMADAQ</td>
<td>Digital Microphone Array Data Acquisition.</td>
</tr>
<tr>
<td>DMAIO</td>
<td>Digital Microphone Input Output.</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier transform.</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array.</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things.</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol.</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network.</td>
</tr>
<tr>
<td>MEMS</td>
<td>MicroElectroMechanical Systems.</td>
</tr>
<tr>
<td>NAH</td>
<td>Near-field Acoustic Holography.</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation.</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol.</td>
</tr>
<tr>
<td>PoE</td>
<td>Power over Ethernet.</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol.</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory.</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture.</td>
</tr>
<tr>
<td>SWOT</td>
<td>Strengths, Weaknesses, Opportunities &amp; Threats.</td>
</tr>
<tr>
<td>UTP</td>
<td>Unshielded Twisted Pair.</td>
</tr>
<tr>
<td>WAN</td>
<td>World Area Network.</td>
</tr>
<tr>
<td>WHO</td>
<td>World Health Organization.</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thermal image of electromotor made on a flir t640. (Flir 2015 [3])</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Original image of power generator. (Sorama 2015)</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Power generator image with sound image overlay. (Sorama 2015)</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Overview Beamforming steering delay summed signal. (R.M.A. van de Looij 2015 [4])</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>(a)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>(b)</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Overview of the Beamforming algorithm steps</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>Overview Holography steps. (R.M.A. van de Looij 2015 [4])</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>Components and flow of the current software architecture design</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>Data files used to store the raw and processed data</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>Single 8X8 microphone grid called: Cam64</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>Full 4X4 module grid which is a 32X32 microphone grid Sound Camera called: Cam1K</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>Components and dataflow of current design</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>Components and flow of new design</td>
<td>9</td>
</tr>
<tr>
<td>15</td>
<td>Overview of Software Architecture patterns</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>(a) Component Based Architecture</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>(b) Event-driven Architecture</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>(c) Service Oriented Architecture</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>Core Concepts Relations</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>Publish/Subscribe model with multiple publishers to two topics with multiple subscribers</td>
<td>21</td>
</tr>
<tr>
<td>18</td>
<td>Rozanski &amp; Woods [34] viewpoints library in respect to the 4+1 view of Kurchten [35]</td>
<td>23</td>
</tr>
<tr>
<td>19</td>
<td>Context view in an UML Context Diagram</td>
<td>24</td>
</tr>
<tr>
<td>20</td>
<td>Functional view in an UML Component Diagram</td>
<td>25</td>
</tr>
<tr>
<td>21</td>
<td>Development view in an UML Model Structure Diagram</td>
<td>27</td>
</tr>
<tr>
<td>22</td>
<td>Deployment view in an UML Deployment Diagram</td>
<td>28</td>
</tr>
<tr>
<td>23</td>
<td>Overview of different type of containers/virtualization</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>(a) Type 1</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>(b) Type 2</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>(c) Type 3</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>Abstract Class diagram proof of concept</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>UML Sequence diagram for doing a measurement in the proof of concept</td>
<td>40</td>
</tr>
<tr>
<td>26</td>
<td>Data flow between multiple serviceBusses</td>
<td>41</td>
</tr>
<tr>
<td>27</td>
<td>UML Sequence diagram for starting synchronized measurement over multiple DMAIO boards</td>
<td>42</td>
</tr>
<tr>
<td>28</td>
<td>Digilent ZedBoard RevA sideA</td>
<td>43</td>
</tr>
<tr>
<td>29</td>
<td>Sorama DMAIO sideA</td>
<td>44</td>
</tr>
<tr>
<td>30</td>
<td>Screenshot of the device table out Sorama Portal with device with id: '#1' is online</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>Screenshot of the console on Sorama Portal in which the device response shows it completed measurement</td>
<td>46</td>
</tr>
<tr>
<td>32</td>
<td>The hologram from measurement #1 on the Portal hologram viewer, at two different selected frequencies bands</td>
<td>47</td>
</tr>
<tr>
<td>Number</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>The hologram from measurement #2 on the Portal hologram viewer, at two different selected frequencies bands.</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Photo of pistonphone fixed on lab stand for a cam64.</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Microphone mapping of Cam64 used during experiments.</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Screenshot of the mockup Portal, which was made for the proof of concept implementation.</td>
<td></td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Summary of connectors on the new integrated acquisition (DMAIO) board</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Overview of the Stakeholders for the software architecture</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Programming Language comparison chart</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>Discovery comparison chart</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>Communication comparison chart</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>Message Broker comparison chart</td>
<td>34</td>
</tr>
<tr>
<td>9</td>
<td>Internal/External language comparison chart</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>Container platform comparison chart</td>
<td>36</td>
</tr>
<tr>
<td>11</td>
<td>SWOT analysis of the designed architecture</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>Specification of the pistonphone used during the experiments</td>
<td>63</td>
</tr>
<tr>
<td>13</td>
<td>Specification of the mockup Portal</td>
<td>64</td>
</tr>
</tbody>
</table>
Listings

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic skeleton of service Object</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>Template for sending messages over the serviceBus</td>
<td>39</td>
</tr>
</tbody>
</table>
1 Introduction

In this section a small introduction is given about Sorama, the company on whose behalf the research reported in this thesis has been conducted. First, some background information is given about the company and a description of a typical use case for the Sound Camera. Next, a brief discuss about the algorithms used for sound analysis offered by Sorama. Then, a discussion about the hardware and software that constitute the current version of the Sound Camera. Once this is done, the context is available to formulate the problem statement this thesis addresses and its derived research goals. Finally, an outline of the remainder of the report is given.

1.1 Context/Company/Application domain/Scenario

1.1.1 Context

The World Health Organization (WHO) \[1\] concluded that noise pollution is linked to a range of health problems. The biggest problem is the loss of sleep. The WHO advises that background noise should be on average 30 decibel and should not exceed 45 decibel for a good night’s sleep. In Europe around 70 million people are exposed to daily vibrations and noise levels of 55 decibels or more from traffic alone.

Noise and vibration awareness needs to play a more prominent role in our daily lives. The mission statement of Sorama addresses this issue and reads as follows: “Sorama intends to be the best company in the world for identifying and reducing unwanted noise in products and surroundings. This way Sorama will add to the quality of life for people wherever they are: at their work, at home or in traffic.” \[2\]

1.1.2 Company

Sorama is a Dutch company located in Eindhoven. Sorama is a spin-off of the TU/e and has been founded in 2009, by Rick Scholte, a former mechanical engineering PhD student. It has developed a Sound Camera which, visualizes sound and vibrations in products, providing unmatched precision, ease of use and accessibility.

Sorama offers several services and solutions in the area of optimization and design of sound and vibrations for product developers, construction, engineering and consultancy companies. These services are in the form of project-based advice and single measurements on problems/projects. To measure and make sound and vibrations visible, Sorama has developed a Sound Camera. This Sound Camera is sold with a membership for their online platform called the Sorama Portal. On the Sorama Portal the measurements can be analyzed for a small amount of credits per action, which are sold separate of the membership. Besides credits, the customer can buy expert tokens. These tokens can be used to asked for help/advise from Sorama.

1.1.3 Application domain

The Sound Camera is used to visualize sound and vibrations in both near and far field with two techniques called Near-field Acoustic Holography (NAH) and Beamforming. To that end, Sorama has developed specialized algorithms to perform these techniques and Sorama provides access to these algorithms only via their online portal, the Sorama Portal.

1.1.4 Scenario

Situations in which the Sound Camera is used are for example, during product development, to make a product quieter. The following scenario describes a typical use case of a Sorama customer.
and his Sound Camera. So assume this customer has designed a new low noise power generator, but discovers that the noise it makes is more than expected. The goal is to improve the excess noise damping of the power generator. Therefore the costumer needs to find the cause/source of the noise. For this goal the costumer uses the Sorama Sound Camera to map the causes/sources of the noise. To do these measurements the following steps need to be taken by the costumer:

1. Set up the Sound Camera and connect it to the PC,
2. Login into the Sorama Portal,
3. Create a new project,
4. Make a picture of the object,
5. Place the Sound Camera in front of the object,
6. Align the Sound Camera with the use of the previous made object picture,
7. Start the power generator,
8. Take a measurement,
9. Repeat Steps 5 till 8 to cover the whole object,
10. Select a specific sound frequency to analyze,
11. Analyze the selected frequency of the measurement.

As can be seen from this scenario, the Sorama Portal is the user interface of the Sound Camera. Moreover, as Step 11 shows the users can analyze their measurements. To protect the intellectual property incorporated by the signal processing algorithms, the sound data is processed online within the Sorama Portal environment and not on the user’s PC. Functions which the user can perform on the Sorama Portal are: Holographic analysis and Beamforming.

During the entire process the user needs to be connected to the Internet to keep the connection with the portal alive. Between Step 10 and 11 the selected frequency data is uploaded to the Sorama Portal for processing. Step 11 can only be done online on the Sorama Portal and is not processed locally.

The results of a measurement are presented in a heatmap. The most well-known application of a heatmap are the images of a thermal camera. Figure 1 shows a heatmap of a generator.
produced by a thermal camera in which red displays a hot surface and blue the opposite, a cold surface. In the sound images generated by the analysis software on the Sorama Portal red stands for a high pressure spot, green stands for an equilibrium and blue for low pressure. For example Figure 3 shows the original image of the object and Figure 4 shows the object with the sound image overlay.

1.2 Algorithms

The main two algorithms used by Sorama for signal analysis are Beamforming and Near-field Acoustic Holography (NAH). Both techniques are described briefly in the following two subsections. These two techniques are sharing two steps that are: the decimation of data performed at the DMADAQ (see Section 1.4.1) and the Fast Fourier transform (FFT) which is done locally at the PC. The mathematical parts of the techniques are considered as black-boxes for this project, because of the mathematic background of sound imaging is outside the scope of this project.

1.2.1 Beamforming

Beamforming is a signal technique to analyze directed signal data. Figure 4 shows that by knowing the steering delay necessary to synchronize the signal from an array of equally spaced microphones such that a single summed signal of high amplitude results, instead of a sequence of low amplitude signals, the direction of the sound source can be determined. The higher the amplitude of the sound indicates a higher probability that that sound originate from the source corresponding to the used steering delay. Figure 5 shows the computation steps of the Beamforming algorithm.
1.2.2 Near-field Acoustic Holography

Near-field Acoustic Holography (NAH) was first introduced in the 1980’s by Williams and Maynard [5]. NAH is a technique to propagate a set of pressure distribution points from one plane to another plane. Figure 6 shows the different stages and the transformations needed to back-propagate [6] the measured sound data from the Sound Camera to the origin of the source. From there it is propagated forward in time with the source as origin for visualisation.

1.2.3 Similarities of the techniques

For both the Beamforming and NAH the first two steps of processing are the same. Currently the Decimator, which turns the bitstream of the microphones into a pressure integer at a specific point in time, is implemented in the FPGA on the DMADAQ of the Sound Camera. More about this data acquisition can be found in Section 1.4.1 where the current hardware-design is explained. Besides the Decimator a second similar processing step needed to be performed, namely the Fast Fourier transform (FFT). The FFT step is used by both algorithms. It turns the time signal into a frequency spectrum. This step is done locally on the client’s PC, before the selected frequency data is uploaded to the Sorama Portal.

1.3 Software

1.3.1 Current Software Architecture

Figure 7 shows a general overview of the software architecture as it is currently implemented. The left side, depicted by the block ‘local’, runs locally on the user his computer, the right side, depicted by ‘Cloud’, runs in the Microsoft Azure Cloud[7]. It is called the Sorama Portal and has been developed in-house. The Silverlight parts needs to be connected to the Cloud to be able to run.
Figure 6: Overview Holography steps. (R.M.A. van de Looij 2015 [4])

Figure 7: Components and flow of the current software architecture design.
Each microphone in the Sound Camera generates a bitstream at 1.5 MHz. Data samples are taken with 1 bit samples. For the Sound Camera is the bitstream converted to a 32 bit data sample which results in a 1.5 Mbits/s datastream per microphone. In terms of pressure samples consisting of 32 bit/samples the sample rate is given by the following equation:

\[
\frac{1500000 \text{bit}}{\text{seconds}} \times \frac{1 \text{sample}}{32 \text{bit}} = 46.875 \text{kHz samples/second}
\]  (1)

For an array of \( X \) microphones the amount of data generated each second \( Y \) is:

\[
Y \text{Mbits/s} = 1.5 \times X
\]  (2)

For example the Cam64 shown in Figure 9 is an 8x8 microphone Sound Camera, this means a total of 64 microphones. The total amount of bits/s is then: \( 1.5 \times 64 = 96 \text{Mbit/s} \)

This means that if the platform has 1GB of memory available, a measurement can last at most \( \frac{1024}{96} = 85.33 \) s before the memory is full.

After each measurement there are two files stored locally on the user’s pc. Figure 8 shows the concept layout of the two data files; the raw data file DataFile which contains for each microphone and sample time the measured pressure. And the frequency spectrum file FrSpFile which is calculated by the FFT based on the raw data.

1.4 Hardware

The overall design of the Sound Camera is designed as flexible as possible. The base of the Sound Camera is a microphone module which is a grid made of 8X8 digital MicroElectroMechanical
Systems (MEMS) microphones. Each microphone is equally spaced, in both horizontal and vertical direction, by two centimeter from its neighbors, as can be seen in Figure 9. This grid can theoretical in design be endless scaled in both the horizontal as vertical direction.

1.4.1 Current Design

Currently Sorama sells two models of the Sound Camera; the 'Cam64' (Figure 9) and the 'Cam1K' (Figure 10). Both Sound Cameras can be equipped with a motion system, to form more versions of the Sound Camera.

The biggest model, the 'Cam1K', consist of four columns that each consist out of four stacked microphone modules of each 64 microphones, this means in total 16 microphone modules which contains in total 1024 MEMS microphones. Each of the four columns is connected via three Unshielded Twisted Pair (UTP) cables to a shared acquisition device called the Digital Microphone Array Data Acquisition (DMADAQ). A dedicated multiplex protocol developed in-house is used over these cables. A maximum of four columns of four modules each can be connected to a single DMADAQ. Multiple DMADAQ’s can be interconnected by a cable to synchronize the triggering of a sound recording. Thus the Sound Camera can be scale in both the horizontal and vertical axis. Besides synchronizing the triggering, this interconnection cable is needed to synchronize the internal clocks of the individual DMADAQ’s. Each DMADAQ still has his own USB2.0 connection with the PC, because only the clock and trigger are synchronized by the interconnection cable and not the data. Figure 9 shows their product called ‘Cam64’ which is a single microphone module, that has a total of 64 MEMS microphones. For the model ‘Cam64’ the same DMADAQ and cables are used as the ‘Cam1K’ model.

Figure 11 shows the individual components of the current hardware design and the data flow between these components. The bitstream generated by the microphones is multiplexed into a single bitstream which is then placed on the UTP to be sent to the DMADAQ on which it is demultiplex back into the single bitstream per microphone and then by the Decimator converted into a pressure integer which is then send via USB2.0 to the users his PC.

The current design has a number of shortcomings, and therefor a new design is required. The first shortcoming is the data integrity on the UTP cables. The proprietary dedicated multiplex communication protocol over the UTP cables is not in line with the specifications of the cable. Therefor a lot of noise is generated by the cable to the extent that sometimes the entire microphone data is lost. The second shortcoming is the hassle generated by the stiffness of the bundle of the UTP cables between the Sound Camera and the DMADAQ. This makes the current design inflexible. The third shortcoming is the limited data transfer rate of USB2.0, therefor real-time readout of the entire 1K model is not possible. Because $1024 \times 1.5 \text{Mbit/s} = 1536 \text{Mbit/s}$, USB2.0 has a maximum throughput of $480 \text{Mbit/s}$.

1.4.2 New Design

To solve the cable problems of the old design and to improve the flexibility of the Sound Camera, a new design is made. Within this new design the DMADAQ is integrated within each column, this means the ‘Cam1K’ has now four separate DMADAQ’s built in. This new version of the DMADAQ which is called the DMAIO, which stands for Digital Microphone Input Output. The DMAIO board contains a Xilinx Z-7010 series chip which contains a dual core ARM processor and has 1GB of DDR3 low-power memory. Table 3 gives a summary of the specifications of the Z-7010 chip. The board contains 128MB of non-volatile Read Only Memory (ROM), this means
that no data can be stored during run-time on the Sound Camera which therefor needs to be stateless in design. Figure 12 shows the separate components and its flow of data or communication between the components. The DMAIO board has the following connections which are summarized in Table 2. The new design can be powered by both the DC Jack connector as well by Power over Ethernet (PoE) which follows the ISO 802.3af standard [10].

As the operating system Sorama has chosen for PetaLinux [11] which is the embedded-Linux solution from Xilinx for the Zynq-7000 series. Xilinx created a full supported build and run environment especially for their chips, including the Zynq-7000. This suit is called PetaLinux [11].

In the new Sound Camera the throughput problem has been solved. To real-time read out the Sound Camera with the new hardware design the USB2.0 throughput shortcoming is solved in two different ways. The first solution is that now one DMAIO read-out a maximum of 256 microphones. This means $256 \times 1.5\, Mbit/s = 384\, Mbit/s$, which is below the stated maximum throughput of $480\, Mbit/s$ [8]. The second solution is the new gigabit network interface. This interface has a maximum throughput of $1000\, Mbit/s$.

### 1.4.3 External trigger

Currently the Sorama Portal supports the use of an external trigger function. With an external trigger the start of a measurement can be externally started/triggered. By using an external
Table 3: Small summary of specifications of Xilinx Zynq-7010[9]
trigger the delay between start and the 'to measure' event can be minimized, because this process is automated. For example when measuring a sunroof of a car, a trigger signal can be used to start the measurement exactly when the mechanical part of the roof starts moving. A sensor is triggered which then starts the measurement on the Sound Camera.

This trigger can be installed in two ways. The first way is by an external I/O board from National Instruments the 'NI USB-6009' 8 Inputs, 14-bits Multifunction I/O which is connected by USB to the clients PC. The second way is by an internal trigger I/O on the DMADAQ. The internal trigger is the most precise way at the moment, because this internal trigger is not influenced by the unknown variable delay introduced by the USB channel.

1.5 Problem statement

The new hardware design described in Section 1.4.2 creates the need for a new software architecture design which can be used with both the new and current version of the Sorama Sound Camera. The main research questions are:

1. "What software architecture is needed to make use of the new hardware version of the Sound Camera?"

2. "How do you design this software architecture and can it be implemented on the new hardware version of the Sound Camera?"

1.6 Goals

The goal of this project is to design and implement a software architecture for the Sound Camera. The smallest design needs to support at least the following scenario: The user can see which Sound Cameras are online and available. The user can select an available Sound Camera and can start a new measurement. From the measurements made in the previous step, it is possible for the user to select his first frequency for analysis. After this analysis, the user must have the possibility to select and analyze a different frequency from the same measurement.

1.7 Outline/Overview

The remainder of this thesis is organized as follows. Section 3 describes three different perspectives; User, Company and technical perspectives. From these perspectives and the Sections 1 and 2 the requirements of the needed software architecture are extracted and formulated in accordance with the S.M.A.R.T. principles. Section 4 describes the new software architecture with the help of viewpoints and perspectives. Section 5 describes the chosen techniques and how the proof of concept implementation of the software architecture is made. Section 6 evaluates if the designed and implemented software architecture meets the requirements. Finally, the report is concluded and future work is described in Section 7.
2 Related Work

A device which can be remotely approached and controlled via the Internet, belongs to what is nowadays referred to as the Internet of Things (IoT). The new Sorama Sound Camera with its ability to connect to the Sorama Portal, the Sorama server environment which is responsible for the computation and analysis of the data acquired by the Sound Camera, can therefor be seen as an IoT device.

IoT is an active research field and here we briefly look at related work which enables us to position the Sound Camera in this field. IoT-research concentrates on a lot of aspects, but here we focus on architecture, nature of devices and the runtime environments. In order to do so we recall the major characteristics of the Sorama Sound Camera. It consist of a small number of modules, that are connected into a Local Area Network (LAN), each of these modules contains a powerful ARM processor and it has no restriction on its power budget. Only when PoE is used the budget is limit, but still not comparable to that of battery power. This makes it possible to perform a small amount of local computation on these modules, which is necessary to reduce the amount of data sent to the Sorama Portal for the final analysis.

In [12] the major architectural challenges in IoT with respect to standardization, privacy, naming and identity are discussed. In [13] a reference architecture for the IoT is presented. Within this model shown in Figure [13] the Sound Camera is made of a number of devices which then consist out of large amount of microphones that can be classified as sensors. These sensors cannot be individual controlled, but a selection of a subset of microphones can been seen as a configuration of the Sound Camera.

In [14] six types of system architectures are described. Figure [14] shows the schema of a type 4 IoT device. The left side, as depicted by 'local', is represented by the Sound Camera. In more detail the 'Device' is represented by the 8x8 microphone module. The 'Resource' is represented by the Z-7010 chip (DMAIO board) and 'Controller service' by the in this research designed software architecture. The right side, as depicted by the 'cloud', is represented by the Sorama Portal. Furthermore the main characteristics of a type 4 IoT device are multiple nodes that can perform local analysis. For the Sound Camera this is represented by multiple DMAIO boards and by the possibility of local processing for example the FFT. The Sorama Sound Camera can be classified as a type 4 system architecture.

The service provided by the Sorama Sound Camera to its users allows them to identify sound and vibration sources on their products which are not according to the product specification. This service is made of a composition of more low level services provided by the Sound Camera. Not all services need to be instantiated on all of the modules, to save resources. A possible candidate is a SOA architecture. In [15] a suggestion for discovery and selection of these low level services is given. [16] answers the question on how to secure these services via OAuth and how to position this in the architecture.

IoT devices vary a lot in both the application domain, in smartness and resource footprint. An early example in the area of home automation is given by [17] which is not very smart but has no restriction in use of power. In [18] an example in the domain construction is given of a not smart and limited power IoT power tool monitoring. For health care a smart bracelet [19] is designed, which is smart and is focused on low power and bandwidth consumption. Usually these devices are embedded devices that offer a limited resource platform for the service they provide. Often a large number of these devices work together to provide a high level service to its users. In this respect the Sorama Sound Camera is somewhat special. Recall that it has both a powerful ARM processor and no restriction on its power consumption in contrary with the examples given.

To recall from Section [1] the choice for Petalinux as operation system was already made.
Figure 13: UML representation of the IoT Domain Model accordingly Enabling Things to Talk\cite{13}.

Figure 14: IoT level 4 accordingly Internet of Things A Hands-On Approach\cite{14}.
by Sorama, but in the field of runtime and cloud environments multiple platforms arose. For example multiple cloud, as depicted the right side in Figure 14 platforms arose 20–22. At the moment of this research one of these platforms 20 was only announced another was in beta 21 and the third platform 22 was just released. For example for the runtime environments on the IoT device, as depicted the left side in Figure 14, 11, 23–25 are examples of IoT related operation systems.

At the level of software architecture a widespread of topics is research. In 26 an approach is given to objectively select a Software as a service Platform. The book 27 is a view on designing, creating and maintaining a software architecture. It defines the term ‘software lifecycle management’ which states that you not only focus on the development, but more at the maintainability of the software.

2.1 Architecture patterns

There are many software architecture patterns, however for this research the focus was on software architectures which allows the software to be built in different blocks/components, which represents the different functionalities that the Sound Camera can offer. The following three software architectures are inspected during this research and will be described in the following subsections: Component Based Architecture, Event-driven Architecture and Service Oriented Architecture (SOA).

2.1.1 Component Based Architecture

In 28 the basics of a Component Based Architecture is described. Figure 15a shows a Component Based Architecture in which each functionality is modeled as a loosely coupled component in the application, which can be easily switched by a different implementation. This is possible because internal communication and actions are predefined.

2.1.2 Event-driven Architecture

In 29 the basics of an Event Driven Architecture is described. Figure 15b shows an Event Driven Architecture. This architecture is based on the Component based Architecture but the difference is that the invocation of components is event driven instead of direct communication between components.

2.1.3 Service Oriented Architecture

In 30 the basics of the Service Oriented Architecture (SOA) is described. Figure 15c shows a SOA design, it similar to the component based architecture as described in Section 2.1.1. In a SOA each component of the application is offered as a service to each other. Communication between the services is typically done over a network. In principle each services are and can be: platform, product and vendor independent.
(a) Component Based Architecture  
(b) Event-driven Architecture  

(c) Service Oriented Architecture  

Figure 15: Overview of Software Architecture patterns.
3 Requirements Analysis

In the requirement analysis, the minimum requirements for a novel software architecture for the Sound Camera are defined. First, an outline of the requirement gathering process is made. After that, the process of using the Sound Camera is presented through three distinct perspectives, namely the User-, Company- and Technology perspective. Finally, the acquired requirements are categorized into groups of user- software- and hardware requirements and defined according to the S.M.A.R.T. method.

3.1 Analysis

Before a software architecture is designed the requirements needs to be formulated. The current Sound Camera is used by two parties; The costumer of Sorama which uses the Sound Camera to solve its sound/vibration problems and the company Sorama itself which develops the Sound Camera and tries to find new markets to deploy the Sound Camera. These parties have their own requirements which needs to be fulfilled by the new software architecture. Therefore two perspectives are created that describe the usages and needs of these two parties which then can be used to extract their requirements.

Besides requirements originating from these two parties there are constraints and requirements which are created by the relation and combination of software and hardware. This means that for this research project three perspectives can be formulated. These perspectives are:

User perspective which describes a situation in which the Sound Camera is used by a costumer of Sorama.

The Company perspective which gives an overview of how Sorama sees the current use of the Sound Camera and the associated usability problems that occur.

The Technical perspective which describes the use of the Sound Camera in its different software conditions; what are the data flows inside the software and hardware and how are those connected.

The perspectives are defined based on internal customer case studies like the feasibility study for Inalfa Roof Systems [31] and several discussions at the beginning of this research with employees of Sorama.

From the three perspectives, the existing software architecture described in Section 1.3.1 and the current and new hardware designs described in Sections 1.4.1 and 1.4.2 the requirements are extracted.

3.2 Perspectives

This section presents three perspectives given from three different angles. The user perspective, the company perspective and the technical perspective.

3.2.1 User perspective

The User perspective describes the typical usage of the Sound Camera by a customer of Sorama:

The general flow during a measurement is as follows: First the product whose sound profile has to be determined is roughly scanned with beamforming, this can be done from a larger distance than NAH. With beamforming the user can find the places of interest to further investigate...
in more detail with NAH. So, after finding the places that generate most of the noise/vibrations, the user starts using NAH to find the exact cause of the problem.

With beamforming, the result of the Sound Camera can be viewed live on the PC of the user. Therefore, the user can freely move the Sound Camera to zoom in and get a more in depth view into places of interest.

For NAH, the Sound Camera needs to be placed close to the location of interest, within the near-field, which is typically three centimeters from the object. A measurement of approximately one up to five seconds is made depending on the user’s settings. The user himself may determine the length of the measurement.

If the user wants to scan an object/place of interest that is bigger than the Sound Camera or if he wants a resolution higher than the default two centimeter, the user can make overlapping or adjacent measurements. With the use of a reference microphone, which needs to be connected during all the separate measurements, the NAH algorithm can stitch the multiple recordings together to get a higher resolution or to analyze a bigger area.

3.2.2 Company perspective

The following perspective describes the current situation and the future perspective of Sorama:

Sorama sells the Sound Camera together with a subscription on their Sorama Portal. This is done to safeguard the proprietary algorithms for Beamforming and NAH which are vital for Sorama. This means a working Internet connection is needed during measurement and analyzes of results. Sorama keeps the price of the Sound Camera as low as possible. To achieve this, their earnings are based on the selling of subscriptions and pay by use of the Sorama Portal. Each action performed by the users that needs to be processed on the Cloud platform will be accounted by credits. Also expert tokens exist; these tokens gives the user the opportunity to ask for knowledge/help by Sorama employee.

Sometimes the customers experience difficulties during their measurement because of poor Internet connections. Besides poor Internet, the question arose if measurements can be made in situations where no Internet connection is available. For example during test trials of a new vessel, customer want to do measurements over a period of a single day or even several weeks. During such a trial, no or poor Internet is available because the vessel is mostly far away from shore. In the current Sorama Portal this is not possible, so a solution must be found to make offline measurement possible and to analyze data in the cloud at a later time, after an Internet connection has become available. This options creates an additional question: How to validate that a measurement went correctly instead of finding out to late when back online. So if possible a solution should be found to offline validate measurements in such a way that a correctness probability can be given how successful the measurement went.

The current hardware design does not have an interface to directly connect the Sound Camera to the network, in stead it requires an intermediate PC to which it connects via an USB2.0 connection. Moreover because of their current choice to use Silverlight on the PC to connect to the Sorama Portal. Therefore it uses an only for Windows developed acquisition client, it requires that the PC runs a Windows operating system, other operating systems like Linux or OSX are not supported. The new hardware design with its new Ethernet interface would make it possible to make the Sound Camera a standalone device. If the Sound Camera is a standalone device Sorama can go to a total Cloud environment on which it does not matter what kind of device you use. For example your Laptop or Tablet, in this scenario the users devices becomes only a terminal to the Sorama Portal. In the ideal situation it should make no difference for the user to change terminals during the process.
3.2.3 Technical perspective

The following perspective describes how the current hardware and software are used:

First, the user connects his model of Sound Camera via USB2.0 to his computer. He opens his web browser and surfs to the Sorama Portal. After he is logged in, the user can create a new or open an existing project. When a new recording is initiated, the software asks what type of Sound Camera is used, the amount of recording time that the user wants, and some parameters of the measured object. After that, a background checklist is performed:

- Is the acquisition-client installed and running?
- Is the Sound Camera connected?
- Is the webcam connected?

If all the items above are valid, the following measurement process is performed in the background; The Sorama Portal gives a ‘go’ to measure and the acquisition-client sends a signal to the DMADAQ to reset itself and prepare to measure. After an ‘ok’ is given back, the acquisition-client gives a ‘go’ to the DMADAQ and tells the Sorama Portal that it is measuring. Now the DMADAQ records for the requested recording time and stores all the preprocessed microphone data in its local memory. After it has recorded the specified amount of time, its sends a ‘done’ signal to the acquisition-client. The acquisition-client downloads the data out of the memory of the DMADAQ and stores the data in the raw data file called: DataFile, which is hosted locally on the user’s computer.

Then the first step of the algorithm, a FFT, is done. This means that the recorded signal in time domain is converted into a spatial frequency domain for each microphone and stored locally in the file called: FrSpFile. The spatial frequency domain data is used to show the user a frequency domain spectrum in which the user can select frequencies to inspect. The selected frequencies are extracted from this local storage and uploaded to the Sorama Portal, where the data is processed and presented to the user for analysis. If the user wants to analyze a not previous selected frequency. This frequency is then again extracted from the local storage and is then again uploaded to the Sorama Portal for processing. If the user wants to reanalyze on a different PC, only the previous selected and uploaded frequencies can be chosen.

3.3 Requirements

In this section all the requirements created by the method described in Section 3.1, all the requirements are split up in to three different categories: Users, Software and Hardware requirements.

The requirements are formulated with the use of the well know mnemonic S.M.A.R.T. Which ensures that all the requirements are tested against the following points: Specific, Measurable, Attainable, Realistic and Time-bounded (S.M.A.R.T.) [32].

3.3.1 User

USE01 The user can do measurements.

USE02 The user can select a frequency to inspect.

USE03 The user can inspect multiple frequencies.

USE04 The user can see a list of his available Sound Cameras.
USE05 The Sound Camera can be operated by a terminal with a web browser installed.
USE06 The user cannot overspend his credit balance.
USE07 The user can control the Sound Camera through the Sorama Portal.
USE08 The user does not need an Internet connection to do measurements.
USE09 The Sound Camera can only be operated by the owner of the Sound Camera.
USE10 The user need to be logged in to use the Sound Camera.

3.3.2 Software
SOF01 The Software Architecture is programming language independent.
SOF02 The Architecture needs to be portable so it can support the current hardware design.
SOF03 The designed and created software for this architecture can be updated remotely.
SOF04 The Sound Camera can do preprocessing of measurement data.
SOF05 The Sound Camera support doing a single measurements for a predefined length of time.
SOF06 The Sound Camera supports doing continuous measurements for an undermined amount of time.
SOF07 The Sound Camera can continuously stream data to an end-point.
SOF08 Communication between the device and the online Sorama Portal is secured.
SOF09 Communication between the linked devices is secured.
SOF10 After the device is powered up, it makes itself known to the Sorama Portal and surrounding devices.
SOF11 The device can be operated without the need of a working Internet connection.
SOF12 The device can make off-line measurements with the use of a local storage.
SOF13 The device can respond to commands given by the online Sorama Portal.

3.3.3 Hardware
HAR01 Multiple devices can be linked together and operated as one for the costumer looking Sound Camera.
HAR02 The device operates without the use of persistent storage.
HAR03 Devices that are linked will have their internal clock synced.
HAR04 The Architecture can run on an ARM processor.
HAR05 The Sound Camera is by design stateless.
HAR06 The Architecture is compatible with both versions of the Sound Camera.
HAR07 The Sound Camera supports local storage.
4 Architecture and design

The new to be designed architecture for the Sound Camera is described in compliance with the ISO/IEC/IEEE 42010[33] standards with the help of the method described in the book of N Rozanski & E Woods[34]. The first section will give a small introduction of the Rozanski & Woods method and will elaborate the relation with the 4+1 view model of Paul Kruchten[35]. The second part will describe the adopted architectural design pattern SOA and will formulate the needed components and services to run the architecture. Then the Rozanski & Woods method is applied: First the stakeholders and their roles and concerns are stated. Then the architecture is described using several viewpoints from the Rozanski & Woods viewpoint library. The following viewpoints are chosen: the Context viewpoint, the Function viewpoint, the Development viewpoint and the Deployment viewpoint. To explain how the non-functional requirements are met the perspectives; Availability and Resilience, Development Resource, Usability and Security are used.

4.1 Software System Architecture

For describing the Software Architecture an Architecture Description (AD) is needed. The ISO/IEC/IEEE 42010 standard[33] defines an AD and specifies the needed requirements for such a description. For this research the templates and guides of Rozanski & Woods[34] is used which is build upon the 42010[33] framework. The method of Rozanski & Woods and the ISO/IEC/IEEE 42010[33] standard suggest instead of making an unreadable, gigantic diagram, which contains all the little details and every possible information about the architecture. The First step is defining the stakeholders of the architecture. The stakeholders are the people involved during development and/or will use the architecture. The second step states that the stakeholders all have their own and mostly different informational needs. These needs are then fulfilled by making multiple different viewpoints which define and describes the different parts of the architecture. A single viewpoint can serve multiple stakeholders. This model is based on the 4+1 architectural view model designed by Paul Kruchten[35] in 1995. The third and final step are the non functional requirements, these are addressed using perspectives.

Figure [16] shows the core concepts of the Rozanski & Woods method and how they relate to each other. By using the Rozanski & Woods method it is possible to apply separation of concerns, by creating different viewpoints and perspectives to provide the needs of the stakeholders.
4.2 Architecture pattern

Sorama wants to have a Sound Camera which is flexible in its applications. They want a Sound Camera that can grow in functionality in the future. In the context of this project the usage of the Sound Camera is restricted to product development by Sorama’s customers. However, in the future Sorama is also looking into applications whereby the Sound Camera is used in, for example, Smart cities. Therefore the architecture needs to be flexible and extensible with future functionalities.

The two patterns which support these kinds of flexibilities are Component based Architecture and Service Oriented Architecture (SOA), as described in Section 2.1. With SOA each service can be platform, product and vendor independent, in contrast with component based in which each component is running as his own process in the same environment and interlinked. This is in compliance with requirement SOF01 and therefore SOA has been chosen.

In a SOA each application of the Sound Camera is built as a composition of separate services. This means that the functionality of the Sound Camera can change at run time by starting and stopping services. Communication between services is needed to control and to operate these services. This communication is made possible by a communication bus and is implemented by using a publish/subscribe model. A publish/subscribe model means that services can subscribe themselves via a topic to this bus. Publishers can then send messages to these subscribers by using the corresponding topics. This means that services only need to know the topics to communicate between each other and not a custom method, making the software architecture flexible and service independent. Figure 17 shows how three publishers and three subscribers that are communicating using two different topics. Publisher 2 shows that it is possible to publish to two topics at the same time in a publish/subscribe model. Subscribers can subscribe to one or multiple topics.
4.3 Selection of default components and services

Section 1.6 describes the minimal needed functionality of the Sound Camera. In order to achieve these goals, the goals are repeated and formulated in the different components needed to be implemented: First the user needs to be able to select a Sound Camera out of a list of active Sound Cameras (Req: USE04), therefore the Discovery Service is created to let the Portal which Sound Cameras are active. Then the user must be able to do a measurement (Req: USE01) which is sampling the microphones. The service which samples the microphone data is called the Sampler Service. Then the user must be able to select a frequency out of the previous sampled data and send this data to the Sorama Portal to be analyzed (Req: USE02). To do this the architecture uses the Uploader Service. To make it possible for the user to control the Sound Camera from the Sorama Portal (Req: USE07) the Communication Service is designed.

These five services are started and controlled by the MainController and internal communication is handled and implemented by the ServiceBus Broker. These two are so called Platform services. The architecture divides the services into two categories:

**Platform services** are the services which are needed to run and provide an environment for the application services to exist.

**Application services** are the services that provide the functionalities which are used by the users during usages of the Sound Camera.

The Platform services are:

- **MainController**
  The MainController is the orchestrator within the architecture, it is responsible for starting and managing the ServiceBus and the services.

- **ServiceBus Broker**
  The ServiceBus is the handler for the publish/subscribe bus and allows internal communication between the services and communication between services spread over multiple boards via the LAN (Section 5.3.1).

To implement the described functionalities the following application services are implemented:

- **Discovery Service**
  The discovery service makes it possible to let the user see his active Sound Cameras on the Sorama Portal.
• **Communication Service**
  The Communication service makes it possible to let the user control the Sound Camera from the Sorama Portal via the Internet.

• **Sampler Service**
  The Sampler Service makes it possible to let the user do measurements.

• **Uploader Service**
  The Uploader service makes it possible to send data from the Sound Camera to the Sorama Portal. For example data of a selected frequency to be analyzed.

• **Streaming Service**
  The Streaming service makes it possible to send real-time Sampler data to a specified endpoint. For example data to be used during the viewing of live beamforming.

### 4.4 Stakeholders

The first step of Rozanski & Woods\[34\] is to determine the Stakeholders of the software architecture, these stakeholders are extracted from the previous Sections 1 and 3. During the design of the software Architecture multiple people are involved and each person represents a group of users with each their different demands, requirements and responsibilities. Table 4 shows the stakeholders during and for this architecture design. Two persons, Rick Scholte and Wouter Ouwens, from inside the organization Sorama, one assessor, Rudolf Mak, from the TU/e and a virtual person which represents the customers of Sorama. The master thesis of T.R.M. Braem \[36\] has been studied and used to describes the customers of Sorama.
Table 4: Overview of the Stakeholders for the software architecture

<table>
<thead>
<tr>
<th>Who</th>
<th>Rick Scholte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Sorama</td>
</tr>
<tr>
<td>Role</td>
<td>Project Manager - Acquirer</td>
</tr>
<tr>
<td>Concerns</td>
<td>The architecture needs to fulfill the client’s wishes and should be future proof.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who</th>
<th>Wouter Ouwens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Sorama</td>
</tr>
<tr>
<td>Role</td>
<td>Project Manager - Developer</td>
</tr>
<tr>
<td>Concerns</td>
<td>Requires the architecture to be practical and functional.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who</th>
<th>Rudolf Mak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>TU/e</td>
</tr>
<tr>
<td>Role</td>
<td>Assessor</td>
</tr>
<tr>
<td>Concerns</td>
<td>Validity of design principle architectural and quality of report.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who</th>
<th>Customer X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Company Y</td>
</tr>
<tr>
<td>Role</td>
<td>Customer - User</td>
</tr>
<tr>
<td>Concerns</td>
<td>The architecture supports the functionality they need to operate the Sound Camera.</td>
</tr>
</tbody>
</table>

4.5 Viewpoints

The second step of Rozanski & Woods\cite{34} is to described the architecture from a number of viewpoints. Figure 18 shows the Viewpoints library of the Rozanski & Woods\cite{34} approach in respect to the 4+1 model of Kurchten\cite{35}. For each view of Kurchten\cite{35} a viewpoint of Rozanski & Woods\cite{34} is chosen. The following viewpoints are chosen and elaborated in the following subsections:
The **Context viewpoint** describes the relationships, dependencies, and interactions between the system and its environment. [34]

The **Functional viewpoint** describes the system’s run time functional elements and their responsibilities, interfaces and primary interactions. [34]

The **Development viewpoint** describes the architecture that supports the software development process. [34]

The **Deployment viewpoint** describes the architecture that supports the software development process. [34]

### 4.5.1 Context viewpoint

According to [34] this viewpoint: "Describes the relationships, dependencies, and interaction between the system and its environment (The people, systems, and external entities with which it interacts)"

This Context viewpoint is meant for every stakeholder defined in Section 4.4, because it creates an uniform image of the positioning of the Sound Camera in the environment.

Figure 19 shows in the center the ‘System being Designed’ which represents the Sound Camera. The system is surrounded by objects that externally affect the system. The lines are commented with what type they are and how they affect the system.

The complete context view diagram exists out of an external sound/vibration source which can be measured by the "System being designed". The Sound Camera is connected to the Sorama Portal. This connection is bidirectional. The connection is used by the Sorama Portal to control the Sound Camera and it is used by the Sound Camera to send measurement data to the Sorama Portal.

![Figure 19: Context view in an UML Context Diagram.](image)

### 4.5.2 Functional viewpoint

According to [34] this viewpoint: "Describes the system’s run time functional elements and their responsibilities, interfaces and primary interactions"
This Functional viewpoint is especially meant for both the Assessor and the Developer stakeholder as defined in Section 4.4. The Assessor can see whether all the functions are satisfied. The Developer can see how the relations are between the components.

Figure 20 shows the functional view created in an UML Component diagram: The view contains all the internal and external components that are involved in the system. This view does not define the exact protocol or type of communication between internal and external components, because this is implementation dependent. The MainController is responsible for both starting the ServiceBusServer and starting/stopping the services. Therefore the ServiceBusService is depending on the MainController. Each service must connect via a Subscriber or Publisher or both to the ServiceBusServer on their own unique topic. This means that the services have an interface dependency on the ServiceBusServer. As can been seen in the view, each internal component has its own corresponding interface with an external component. The interfaces with dotted lines on the right-hand side of the CommunicationService and the other components, are not there to be physically implemented, but are only to show the dependency between the CommunicationService and the corresponding component. For example: the CommunicationService sends a command to the SamplerService to start sampling microphone data. In the system this is done by sending the command from the CommunicationService to the ServiceBusServer with the topic ‘Sampler’.

Figure 20: Functional view in an UML Component Diagram.
4.5.3 Development viewpoint

According to [34] this viewpoint: "Describes the architecture that supports the software development process."

This Development viewpoint is meant for both the Developer and the Acquirer as defined in Section 4.4. The Developer can see which supporting elements are available and can be used by the architecture. The Acquirer can see which components need to be obtained and bought.

Figure 21 shows the development view which contains the supporting elements which are needed and can be used to develop and run the software architecture. As can been seen in the Figure the architecture application is built and depending on four different layers:

The **Application layer** contains needed the services provided the different functionalities of the Sound Camera. To recall the services categorized as 'Application services' defined in Section 4.3.

The **Domain layer** contains the components needed to initialize the application layer components. To recall the services categorized as 'Platform services' defined in Section 4.3.

The **Utility layer** contains the libraries shared by the basic development tool kits of the chosen programming languages. For example the '.NET shared libraries' when C# is chosen as the main programming language as described in the coming Section: 5.1.1.

The **Platform layer** contains the platform dependent elements. For example the 'Mono Run-time' which is the Linux implementation of the C# CLR.

Each layer has a dependency on the layer below itself. Both the application and domain layers are using the Utilities directly as can be seen in Figure 21. It is also possible for a component in the application or domain layer to directly access a platform component for more an efficient data flow or because it does not need a pre-made shared library. For example the **SamplerService** which samples the microphones, it would be inefficient to first go through three layers to access the driver. For the domain layer an example is the **ServiceBusServer** which directly connects to the ZeroMQ Library. These specific libraries are deepening on the choices made during the evaluation of programs, these choices are still to be describe and can be found in the coming Section 5.1.
4.5.4 Deployment viewpoint

According to [34] this viewpoint: "Describes the environment into which the system will be deployed and the dependencies that the system has on elements of it”

This Deployment viewpoint is intended for the Developer stakeholder as defined in Section 4.4. The Developer can see how the different components need to be deployed and how the multiple systems are connected.

Figure 22 shows where all the components of the system are deployed. The Figure shows how two physical different systems (DMAIO boards) called 'processingNode’ can be interconnected. The 'processingNode’ contains an ARM as its CPU and usage Petalinux as its operation system. The connection is made by the ServiceBus via the 'MessageBus’ to make it act as one whole system. When two 'processingNodes’ are connected, there are a number components which need to be initiated on both or on one of the 'processingNodes”. On both the 'processingNodes’ are the following components required; MainController and the ServiceBusServer, to make an interconnection possible. To have a functional Sound Camera, all the 'processingNodes’ need to have the SamplerService initialized because this component uses the local file I/O to sample the microphones. For the Sound Camera model 'Cam1K’ this means that four DMAIO boards are connected to form one Sound Camera.
4.6 Perspectives

The third step are the Perspectives that are introduced by Rozanski & Woods[34] for describing the nonfunctional requirements of the architecture. Rozanski & Woods[34] introduces the terminology Perspectives for addressing these architectural points. The perspectives used to describe the new software architecture are:

The Availability and Resilience perspective which shows how to control the fault handling of the system.

The Development Resources perspective which shows how to ensure the system can be designed, build and operated.

The Usability perspective which shows how usable the system is for the user.

The Security perspective which shows how the system controls and maintains security during its lifetime.

4.6.1 Availability and Resilience perspective

According to [34] this perspective purpose is: "The ability of the system to be fully or partly operational as and when required and to effectively handle failures that could affect system avail-
This perspective is intended for the Developer, Acquirer and Assessor stakeholders as defined in Section 4.4. It can be used to determine the stability and fault handling of the system.

The architecture is designed to be able to self-configure and then operate stand-alone or in a configuration/collaboration of a number of DMAIO boards, also mentioned by the requirement HAR01. These boards can find each other with the help of a discovery service or after communicating with the Sorama Portal to load a configuration file which tells the boards about each other. The Sound Camera hardware is designed to be stateless as explained within Section 1.4.2 and stated by requirement HAR05. With each bootup or reboot of the system it starts in the same first initial state. If there is a system failure, a simple option is to reboot the Sound Camera to reset the architecture.

4.6.2 Development Resource perspective

According to [34] this perspective purpose is: "The ability of the system to be designed, build, deployed, and operated within known constraints related to people, budget, time and materials."

This perspective is intended for the Developer, Acquirer and Assessor stakeholders as defined in Section 4.4. It can be used to determine the design of deployment in context to the non-function requirements.

One of the requirements SOF03 requires that it would be possible to remotely update the Sound Camera. In Section 1.4.2 was describe that Petalinux was chosen as the main operating system. Petalinux has a build-in update functionality [37], which allows Sorama to completely update the entire operating system, which includes the software architecture. For executing and controlling how to enroll updates, a service called UpdateService can be implemented which can check against an online repository operated by the Sorama Portal to know when to update. During evaluation of the software packages the requirement that it needs to run on an embedded system is controlled, therefore HAR04 is satisfied. The choice of a SOA pattern allowing services to be build independent of each other, satisfies the requirement SOF01. As will be explained in Section 5.1.1 C# is chosen as the main programming language, which allows Sorama to reuse most of their current code base and will result in a shorter development time.

4.6.3 Usability perspective

According to [34] this perspective purpose is: "The ease with which people who interact with the system can work effectively."

This perspective is intended for the User, Acquirer and Assessor as defined in Section 4.4. They can use this perspective to determine how the usability is of the Sound Camera compared to their nonfunctional requirements.

The new architecture satisfies the requirement USE07 allowing the user to control the Sound Camera via the Sorama Portal without the current acquisition client installed on his PC. As described in Section 3.2.2 the new Sorama Portal will be developed as a HTML5 application. Allowing the user to control the Sound Camera via a device that can run a web browser, which is required by requirement USE05. Requirement USE06 defines that the user cannot overspend his credits, currently the users is only debited when data is processed on the Sorama Portal. This
means not overspending the user his credit balance can therefore be guaranteed on the server side. To be able to change this in the future a dedicated credit service can be implemented. The credit service should be heavily encrypted and should monitor each action performed on the Sound Camera, to know if credits need to be settled. To allow offline measurements as required by requirement [SOFT1] and [SOFT2] a local storage acquisition client can be built that runs on the user’s pc, which can temporarily store the measurement data until an Internet connection becomes available to upload the data to be analyzed.

4.6.4 Security perspective

According to [34] this perspective purpose is: "The ability of the system to reliably control, monitor, and audit who can perform what actions on what resources and to detect and recover from failures in security mechanisms"

This perspective is intended for every stakeholder defined in Section 4.4. They can use this perspective to determine how the systems handles the sensitive sample data securely.

The Sound Camera is used within the R&D compartments of Sorama customers. The disclosure of data of a new product in development is a nightmare for them. Therefore access control and prevention of disclosure of data that can be valuable for competitors or thieves need to be controlled. The fact that the Sound Camera is stateless in design means that no data can be left behind on the system. The data can be uploaded to the Sorama Portal and stored there securely. The fact that there can be no data left on the Sound Camera means that it cannot be obtained via the Sound Camera. To prevent unintended access to the Sound Camera all the communication lines are needed to be encrypted in such a way that both the Sound Camera and the Sorama Portal can verify the identity of the sender. Besides verifying the sender, the encryption is also used that only the intended receiver can read the messages. If a breach in identity is detected the Sound Camera should stop working and should restart. The reboot should remove all sensitive data from the Sound Camera and stop the breach. Each Sound Camera has an unique identification code programmed within its hardware. Each time the Sound Camera is started up the configuration file is loaded from the Sorama Portal, within this configuration file also access control to the services is registered. By reloading this configuration file each time the Sound Camera is started, this means it is guaranteed that the configuration is up to date.
5 Implementation

In this section the implementation of the software architecture is described. First the selection of the used technologies and their implementation is discussed. Then the schematics and the code base of the proof of concept implementation is explained. After the code base is explained some special use cases are clarified on how they are or would be implemented within the software architecture. At the end of this section the used development board is described.

5.1 Selection of implementation

In this subsection the various technologies for building the proof of concept are selected and motivated: As the general programming language C# has been chosen. For the two discovery scenarios the technologies mDNS (Bonjour, Avahi) and an implementation based on the HTTP(S) protocol has been chosen. For the serviceBus a ZeroMQ implementation is made. For the communication between the Sound Camera and the Sorama Portal websockets are chosen and for the wire protocol of the architecture JSON is chosen. During the selection of the technologies the security aspect as described in Section 4.6.4 is always taken into consideration.

5.1.1 Programming Language

Every application service can be developed in its own programming language, but it is recommended to choose one common programming language for the platform services; the MainController and ServiceBusServer.

Table 5 shows a list of candidates and evaluates them according to a number of criteria. These criteria are:

- **Type**: At which moment is the code made platform depending? Runtime, if there exist an intermediate platform independent compiled code. Compile time, if it is generated on compilation of the application.
- **CodeBase**: How much of the current code of Sorama is written in this particular language and can be reused if this programming language is chosen?
- **Knowledge**: How much knowledge do Sorama Employees have of the programming language?
- **Platform**: How much is the programming language designed for embedded ARM platforms?
- **Standard Libraries**: How many libraries are there available in the basic development tooling?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>CodeBase</th>
<th>Knowledge</th>
<th>Platform</th>
<th>Standard Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Runtime</td>
<td>-</td>
<td>+</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>C#</td>
<td>Runtime</td>
<td>++++</td>
<td>+++</td>
<td>++</td>
<td>+++</td>
</tr>
<tr>
<td>C++</td>
<td>Compile</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>++</td>
</tr>
<tr>
<td>C</td>
<td>Compile</td>
<td>+</td>
<td>+++</td>
<td>++++</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 5: Programming Language comparison chart

Within Sorama a large amount of the current control CodeBase is written in C# and the algorithms are written in C++. Therefor the most knowledge and experience are in these languages. The shared libraries given by the .NET framework available for C# can be used for building the
main services components that orchestrate the system. Besides C# is an intermediate language, therefor the created code is cross-platform, which is required by Requirement SOF02 to better support multi-platform. Therefore as the main programming language C# has been chosen. In Appendix C.1 the instructions on how to install mono the CLR for C# can be found.

5.1.2 Discovery Service

The discovery service is required to find active DMAIO boards. There are two scenarios for which this is needed. The first scenario is concerned with the internal connection of multiple DMAIO boards. The second scenario is concerned with discovery by the Sorama Portal to know which Sound Camera is online and ready to use.

The first scenario is done in the Local Area Network (LAN) at which the Sound Camera is connected. This network may vary widely. Therefore a simple discovery technique is needed which does not require the Sound Camera to have any knowledge of the network at which it is connected.

The second scenario is more complex because the Sorama Portal needs to know which of the Sound Cameras are online. Since the Sound Camera is connected to a LAN this makes it impossible to connect directly from the Sorama Portal (WAN) to the Sound Camera (LAN). The Sorama Portal has a fixed address which can be configured on the Sound Camera, so the Sound Camera can advertise itself to the Sorama Portal to overcome this problem. So the Sound Camera connects to the Sorama Portal, this connection usually needs to cross a Network Address Translation (NAT) at the customer side. The NAT is used to translate an internal IP address into in this case an Internet IP address. Normally outgoing connection are not blocked, incoming connections are difficult because the internal IP is unknown by default for the Sorama Portal.

Table 6 shows a comparison of candidate techniques that can be used for discovery. The techniques are inspected on the following criteria:

**NAT-compliant:** Is the technique made to overcome a NAT or Firewall in the design?

**Config:** How much pre-/post config effort is needed to get the technique working?

<table>
<thead>
<tr>
<th>Name</th>
<th>NAT-compliant</th>
<th>Config</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP(S)</td>
<td>+</td>
<td>Know host</td>
</tr>
<tr>
<td>mDNS (Bonjour,Avahi)</td>
<td>-</td>
<td>Zeroconf</td>
</tr>
<tr>
<td>UPnP</td>
<td>-</td>
<td>Zeroconf</td>
</tr>
<tr>
<td>JINI</td>
<td>-</td>
<td>Local Lookup</td>
</tr>
</tbody>
</table>

Table 6: Discovery comparison chart

For internal discovery the best candidates are mDNS (Bonjour,Avahi) and UPnP. The paper of Palmila[38] compares these two techniques, in which the paper concludes that mDNS (Bonjour,Avahi) is better for LAN networks and UPnP more for propriety sensor networks. Therefore for internal discovery a mDNS (Bonjour,Avahi) discovery implementation is chosen. In Appendix C.3 the instructions on how to install Avahi can be found.

For the discovery to the Sorama Portal only one candidate is suitable, because the others do not work from LAN to WAN. Therefore for discovery to the Sorama Portal an implementation that uses the HTTP(S) protocol to make itself known to the Sorama Portal. This is the standard client-server approach where the initiative is on the client side. Therefore this client-server implementation based on the HTTP(s) protocol is chosen.
5.1.3 Communication Service

The communication service is needed to send commands to the Sound Camera and receive the replies from the Sound Camera. Internal communication between services and boards is done via the ServiceBus. This means that the communication service is only responsible for communication to and from the Sorama Portal.

Table 7 shows a comparison of candidate techniques which can be used for communication to the Sorama Portal. The candidates are inspected on the following criteria:

**Push:** Can the technique push messages from the server to the client?

**Pull:** Does the technique requires the client to pull messages from the server?

**NAT:** Is the technique made to overcome a NAT or Firewall in design?

**Implementation:** How large is the implementation effort?

**Throughput:** Is the technique ready for high level of throughput of data?

**Security:** Does the technique have default security options built-in, for example, encryption of messages?

<table>
<thead>
<tr>
<th>Name</th>
<th>Push</th>
<th>Pull</th>
<th>NAT</th>
<th>Implementation</th>
<th>Throughput</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++++</td>
<td>++</td>
<td>yes</td>
</tr>
<tr>
<td>REST</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>+++</td>
<td>yes</td>
</tr>
<tr>
<td>WebSocket</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>++++</td>
<td>+++</td>
<td>yes</td>
</tr>
<tr>
<td>Socket</td>
<td>+</td>
<td>+</td>
<td>+/-</td>
<td>+++</td>
<td>++++</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 7: Communication comparison chart

For communication between the Sound Camera and the Sorama Portal it is needed to be able to cross a NAT, all the compared candidates support this. But Socket mostly requires some effort from the customer’s IT department, they need to make a firewall exception. Therefor the selection is done on Throughput and Implementation, this results in the following techniques WebSocket and Socket. WebSocket is a layer built on top of a Socket which makes it easier to implement. Websockets has a security layer build in [39], this can be used in order to meet the requirement: SOF09. Therefor for communication between Sound Camera and the Sorama Portal the technique WebSocket has been chosen.

5.1.4 ServiceBus

The serviceBus is used to facilitate a publisher/subscribe communication bus for internal services. Also the serviceBus will be used for linking multiple DMAIO boards together to form different models of the Sound Camera. For example the model ‘cam1K’ exist out of four DMAIO boards.

Table 8 shows a comparison of candidate techniques which can be used for a publisher/subscribe bus. The techniques are inspected on the following criteria:

**Type:** What kind of software is it; is it only a library(Lib) or an entire program(Prog) suite?

**Persistent Storage:** Can the technique store messages within its system?
**Overhead:** What is the level of overhead compared to what is needed?

**Language Bindings:** How easy is it to use within different programming languages?

**Security:** Does the technique have default security options built-in, for example, encryption of messages?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Persistent Storage</th>
<th>Overhead</th>
<th>Language Bindings</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCF</td>
<td>Lib</td>
<td>No</td>
<td>++</td>
<td>Only C# supported</td>
<td>no</td>
</tr>
<tr>
<td>ActiveMQ</td>
<td>Prog</td>
<td>Yes</td>
<td>++++</td>
<td>Most Common</td>
<td>yes</td>
</tr>
<tr>
<td>ServiceMix</td>
<td>Prog</td>
<td>Yes</td>
<td>+++</td>
<td>Most Common</td>
<td>yes</td>
</tr>
<tr>
<td>ZeroMQ</td>
<td>Lib</td>
<td>No</td>
<td>+</td>
<td>Most Common</td>
<td>yes</td>
</tr>
<tr>
<td>RabbitMQ</td>
<td>Prog</td>
<td>Yes</td>
<td>++</td>
<td>Most Common</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 8: Message Broker comparison chart

The DMAIO board is an embedded platform. A low memory footprint is important on embedded platforms and also limited performance is available. This means that having low overhead is very important. Therefore a library is the best choice, this results in two remaining candidates: WCF and ZeroMQ. Requirements: \[\text{SOF01}\] states that multi language support is important therefore WCF is not chosen. ZeroMQ is only a set of libraries and can be completely customized to each situation. ZeroMQ has a built-in function which allows encryption of all messages. The encryption functionality can be used to meet the requirement \[\text{SOF09}\]. Therefore for the serviceBus the library ZeroMQ has been chosen. In Appendix C.2 the instructions on how to install ZeroMQ can be found.

### 5.1.5 Wire protocol

To use the serviceBus optimal it is wise to decide for an uniform and distinct wire protocol for the entire system. This wire protocol is used for both internal as external communication, to makes the system easier to extend or to connect to it.

Table 9 shows the different candidates which can be used for the wire protocol. The techniques are inspected on the following criteria:

**Performance:** Is it an efficient language to use on embedded systems?

**Flexible:** How easy is it to extend or to change documents?

**Support:** How big is the support for the wire protocol?

**Binary:** Does the language support binary transport?

**C# support:** Does the language C# supports the wire protocol?
The most known and suitable candidates are XML and JSON. XML has some performance disadvantages over JSON that is shown in the case study done at Montana State University\cite{41}. Therefore as the wire protocol JSON has been chosen.

5.1.6 Software as a Service

To have more control over the memory and processing usage of individual services, or to create a more fault tolerance and isolated environment for the services the architecture could run the services in individual containers or within virtual machines. Figure 23 shows the different type of containers/virtualization. The higher the type the more isolation is created. Type 1 let the services/components run in different unique processes, but will run within and be managed by the same host application and have knowledge of each others existence. Type 2 will create isolation by using Linux namespaces so called containers within the host OS instead of in an application. Type 2 even allow binaries of libraries to be unique between containers. Type 3 is the highest level of virtualization, then even the hardware level is virtualized and unique per application. A type 3 virtualization mechanism is also called a virtual machine.

Table 10 shows the different virtualization techniques that are inspected. The techniques are inspected on the following criteria:

**Language:** Which programing language does the system support?

**Type:** Type1: It is in the program, Type2: It runs in a virtual OS but shared hardware, Type3: Both Hardware and OS are virtual.

**Effort:** What is the relative effort to implement?

**Adaptability:** To what extent is it possible to change the behavior to an application’s needs?

**Own environment:** Is it possible to have all the components running in a Sorama controlled environment?

**Binding:** Are their bindings to control the services from a different process?

<table>
<thead>
<tr>
<th>Name</th>
<th>Performance</th>
<th>Flexible</th>
<th>Support</th>
<th>Binary</th>
<th>C# support</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td>+++</td>
<td>++</td>
<td>++++</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>XML</td>
<td>+</td>
<td>++</td>
<td>++++</td>
<td>+/-</td>
<td>+</td>
</tr>
<tr>
<td>Plain</td>
<td>++++</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+/-</td>
</tr>
<tr>
<td>YAML</td>
<td>++</td>
<td>++</td>
<td>+++</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>GPB</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>HTML</td>
<td>+</td>
<td>+</td>
<td>+++</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>CSV</td>
<td>+</td>
<td>+</td>
<td>+++</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 9: Internal/External language comparison chart
Figure 23: Overview of different type of containers/virtualization.

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Type</th>
<th>Effort</th>
<th>Adaptability</th>
<th>Own environment</th>
<th>Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSGI</td>
<td>Java</td>
<td>1</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Docker</td>
<td>Independent</td>
<td>2</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>LXC</td>
<td>Independent</td>
<td>2</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Namespaces</td>
<td>Independent</td>
<td>2</td>
<td>++++</td>
<td>+++</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Vmware</td>
<td>Independent</td>
<td>3</td>
<td>++++</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 10: Container platform comparison chart

No specific choice is made for the software as a service technique. Because of the high amount of integration and development time to implement these techniques on the DMAIO boards, it is decided to move this to future work. This means that in this proof of concept the services will run within the same Linux namespace. OSGI is no option because the lack of the multi-language support. It supports only Java. Vmware is not option for embedded platforms, because of the overhead generated by the hardware virtualization. Possible candidates are Docker and LXC. With Docker the system is bound to use their environment, and cannot be self-hosted and therefore safeguarded by Sorama itself. So this research suggests that if Sorama wants to do future research in containerizing the services they should use LXC as the virtualization environment.

5.2 Design of proof of concept

In this section the design and creation of the proof of concept is explained. First an overview of the most important classes of the total class diagram is presented. Next, the classes and interfaces used in the implementation are explained. Finally, two code snippets are presented. The first code snippet illustrates a service skeleton and the second code snippet illustrates how to publish a message.

5.2.1 Design principles

Before the proof of concept implementation is explained some overall considerations, choices and design principles are explained. These points are used during the entire development of the proof of concept and therefore explained beforehand.
The following overall considerations, choices and design principles are used in the proof of concept implementation:

**Service as interface:** Each service defined in Section 4.3 is represented as an interface and is running in its own process within the whole proof of concept program instead of being standalone.

**NuGet package manager:** NuGet[^42] is used to find pre-made implementations for, e.g., a JSON parser.

**Inversion of control:** The Inversion of control design principle is used to build an implementation independent system. This is done by implementing interfaces for each system and application service.

**Dependency injection:** This is used to resolve dependencies within the implementation. To be specific the .NET Windsor Castle[^43] implementation is used.

### 5.2.2 Class diagram

Figure 24 shows an overview of the most imported classes of the total class diagram of the proof concept. In this overview only the interfaces and their dependencies are shown and not the actual implemented classes. As showed in the figure, each service is depending on the `MainController`, because the `MainController` will start/stop the service. Services may also interface with the `IServiceBusSubscriber` or `IServiceBusPublisher` if they need to connect to the `serviceBus` for respectively the subscribe or to publish or both. A complete class diagram of the proof of concept can be found in Appendix A.

![Class diagram](image)

**Figure 24:** Abstract Class diagram proof of concept.

### 5.2.3 Listing of interfaces

The program is made in in the language C# using the .NET4.0 Framework. Each type of service is represented by its own interface. The corresponding interfaces are as follows:
To give a service access to the service bus the following interface are supplied:

- ISampler
- IDiscovery
- IUploader
- ICommunication

To let the program know which parts to auto start and which parts needs garbage collection on shutdown, the following interfaces are supplied:

- ISubscription
- IPublishing
- IServiceBusSubscriber
- IServiceBusPublisher

Settings are distributed via a settings Interface:

- ISettingsProperty
- ZeroMQSettingsProperty

The program depends on the following third party libraries which are acquired via NuGet packages manager:

- WindsorCastle v3.3.0
- Newtonsoft.Json v6.0.8
- WebsocketSharp v1.0.3 RC8
- ZeroMQ v4.1.0.17

5.2.4 Code snippet

In this section two code snippets will be shown. The following two snippets will be given:

**Class template:** A basic skeleton of a service Object.

**Publishing message:** how to publish messages via the `serviceBus` to a specified topic.

Listing 1 shows the skeleton class for a service Object. Via dependency injection the Class can receive its dependency via its constructor, in the given example both the `serviceBusPublisher` and `serviceBusSubscriber` are listed in the constructor. This method of injection is supplied and enforced by .NET Windsor Castle. If the object wants to subscribe to a topic, it needs to extend the `IPublishing` which requires the implementation of the method ‘Publish’. This method is invoke when a message is received on his subscribe topic. In this example it is subscribe to the topic: ‘Sampler’.

Listing 2 shows how a service can publish a messages to the `serviceBus`. In this example the text "Measurement Start" is send to the `Communicator`. 
public class MockSampler : ISampler, IPublishing
{
    private readonly ISettingsProperty _settingsProperty;
    private readonly IServiceBusPublisher _serviceBusPub;
    private readonly IServiceBusSubscriber _serviceBusSub;

    // Dependency Injection via the constructor
    public MockSampler(ISettingsProperty settingsProperty,
                        IServiceBusPublisher serviceBusPub,
                        IServiceBusSubscriber serviceBusSub)
    {
        _settingsProperty = settingsProperty;
        _serviceBusPub = serviceBusPub;
        _serviceBusSub = serviceBusSub;
        _serviceBusSub.Subscribe(this, "Sampler");
    }

    public void Publish(Message e, string serviceName)
    {
        if (e != null)
        {
            if (e.EventData == "Record")
            {
                StartSampling();
            }
        }
    }
}

Listing 1: Basic skeleton of service Object

Message msg = new Message
{
    ServiceName = "Communicator",
    EventData = "Measurement_Start"
};

_serviceBusPub.Publish(msg, "Communicator");

Listing 2: Template for sending messages over the serviceBus

5.2.5 Sequence diagram

Figure 25 shows an UML sequence diagram for doing a measurement. It shows all the commands
from startup until doing a measurement and uploading the results to the Sorama Portal. Four
different services, namely the DiscoveryService, CommunicationService, SamplerService and Up-
loaderService are used during this activity. These services are initialized by the MainController
and communication is handled by the ServiceBus.

39
Figure 25: UML Sequence diagram for doing a measurement in the proof of concept.
5.3 Use cases

In this section some special use cases supported by the architecture are explained in more detail. It is shown how they are handled and implemented by the software architecture. The following use cases are explained in the next subsections:

**Connecting multiple ServiceBusses**: How can multiple DMAIO boards be interconnected?

**Doing Measurements**: How can we do a measurement over multiple DMAIO boards?

**Synchronizing Clock**: How can we synchronize the internal clock of multiple DMAIO boards?

**Making offline measurements**: How can we make offline measurements possible?

**Deviating communication**: How can we facilitate for example more demanding communication?

5.3.1 Connecting multiple ServiceBusses

By using a local Zeroconf discovery the DMAIO boards can find each other. After discovery they then can connect each others ZeroMQ Pub/Sub servers. ZeroMQ has the ability to connect two or more Pub/Sub server to each other, when they are connected the messages are forward between the servers.

Figure 26 shows the dataflow between multiple serviceBusses when they are connected.

5.3.2 Doing Measurements

It is assumed that upon start of a measurement the internal clocks of the single or multiple DMAIO board(s) are synchronized with either the Sorama Portal or each other.

The first step is that the Portal sends a 'GetReady' signal to a single or multiple DMAIO boards. Now the DMAIO board(s) can prepare them self(s) to be ready to Sample data. Then after the Portal have received all the 'Ready' signals back, it sends a signal 'StartSampling'. With this command the Portal adds a time variable at which to start sampling. Be putting this variable for example one second in the future, potential delays introduced by the communication channel can be filtered out. Figure 27 shows an UML sequence diagram how the above explained scenario works.
5.3.3 Synchronizing Clock

Synchronizing the internal clock of multiple DMAIO boards can be done in multiple ways:

**SMA port:** By using the internal SMA ports and sending a ‘Worldclock’ signal over it. Accuracy in the nanosecond range.

**PTP protocol [47]:** By using Precision Time Protocol (PTP) that is designed to synchronize clocks of a network connection. Accuracy in the sub-microsecond range [47].

**NTP protocol [48]:** By using Network Time Protocol (NTP) the internal clock can be sync against a big cluster of timeservers available via the Internet. Maintain time to within tens of milliseconds [48].

For each different method a service can be built, lets name this service `clockSync`. Now the architecture can choose the most suitable service corresponding to that situation. The service needs to allow other services to ask for an internal clock synchronization. For example in the use case in Section 5.3.2, the `SamplerService` could ask the `clockSync` service to synchronize the internal clocks of the participating DMAIO boards at the ‘GetReady’ signal.

5.3.4 Making offline measurements

The Sound Camera does not have persistent storage. Therefore the measurement data cannot be stored on the Sound Camera, but need to be sent to the Sorama Portal for long term storage. The first option would be creating a program that facilitates a virtual disk for the Sound Camera that can run on the users PC. This program will pretend to be the available Sorama Portal. To establish a connection between the PC and the DMAIO board or boards, the Zeroconf discovery can be used. Then messages can be send either via the `serviceBus` or by redirecting the `communicationService` directly to the PC. The second option would be designing a `storageService` which can detect if there is an USB thumbdrive installed in the Sound Camera. It is then also possible to run this service only on one DMAIO board, and send the data via the `serviceBus`. Now the user only need to plug one USB thumbdrive in the USB connector of the Sound Camera. But for that solution one remark must be made. It is not possible to do measurements longer than the available memory as explained in Section 1.3.2. A longer measurement requires
real-time storing the data on the thumbdrive of a whole column or on Sound Camera what is not possible, because of the USB2.0 speed limitation as explained in Section 1.4.1.

5.3.5 Deviating communication

Some services could require more demanding communication than the chosen serviceBus can handle. For example a functionality that can benefit from this feature is the live Beamformer. The data between the SamplerService and StreamingService can be more optimal via a direct connection. Therefor the following solution is suggested: Not all service to service communication need to be handled by the serviceBus. Services can negotiate a direct communication method between two services over the serviceBus. This releases services from restrictions on their used communication, thus making the software architecture more flexible in design.

5.4 Development board

During this project the following two development boards are used. Figure 28 shows the Digilent Zedboard. The Zedboard is the Xilinix reference design development board for the Zinq 7000 series. Figure 29 shows the final design of the DMAIO board commissioned by Sorama that will be used in the Sound Camera.

Both development boards are running Petalinux v2014.4. To be able to run the proof of concept, the following default Petalinux RootFS options need to be enabled:

- glib-2.0 (For libc)
- Dropbear (For SSH access)

![Figure 28: Digilent ZedBoard RevA sideA.](image-url)
Figure 29: Sorama DMAIO sideA.
6 Validation & Evaluation

To validate and evaluate this research three phases of testing are done. The first phase, the system, is tested to ensure a correct functionality of the systems as a whole. This is done through the executing of a series of experiments that are build on each other’s findings. The second phase, the components, are checked and validated against the individual requirements that were stated in Section 3.3. The third phase, the architecture, is evaluated using a SWOT to determine the Strengths, Weaknesses, Opportunities & Threats of the architecture as a whole.

6.1 Experiments

To evaluate the proof of concept implementation some experiments are done. These experiments are used to verify whether all the goals as specified in Section 1.6 are met.

The following experiments have been carried out:

Experiment Discovery: Get a list of available Sound Cameras.

Experiment Measurement: Do a measurement on the Sound Camera.

Experiment Analyze: View the results on the Web Portal.

Experiment Repeatability: Ensure the above described experiments are repeatable.

Appendix B shows the setup of the proof of concept used during the experiments, it contains the used hardware and the environment in which the experiments are conducted.

6.1.1 Experiment Discovery

The purpose of this experiment is to demonstrate that the discovery of the Sound Cameras is working, this means that it should be possible to see which Sound Cameras are online and available. The setup of the experiment consists of two Sound Cameras that are registered to a demo Portal, both Sound Cameras are turned off at the start of the experiment. The Portal as shown in Appendix B shows a list called the device table containing both Sound Cameras and their corresponding status which can be online, represented by a row color green, or offline, represented by a row color red.

For the experiment the first Sound Camera, which is registered under the id ‘#1’, will be turned on and will try to connect to the Portal, and should appear to be online in the device table in the Portal. Figure 30 shows that when the Sound Camera identified with the id ‘#1’ is started it turned green in the table on the Portal. So the experiment was successful.

![Figure 30: Screenshot of the device table out Sorama Portal with device with id: ‘#1’ is online](image-url)
6.1.2 Experiment Measurement

The purpose of this experiment is to demonstrate that it is possible to do a measurement on a by the user selected Sound Camera from the list of his active Sound Cameras on the Portal. The previous experiment showed that there is an active Sound Camera. This active Sound Camera is used during this experiment. To make it possible to verify that the measurement is successful a pistonphone of 1 kHz is fixed at, with the help of a lab stand, a known microphone of the Sound Camera as explained in Appendix B. For this experiment the position of the pistonphone is on microphone 14, according the mapping as explained in Appendix B.

For the experiment the active Sound Camera with the id '#1' is triggered from the Portal to do a measurement. The response of this trigger can been seen in Figure 31 from the Portal console which shows that the Sound Camera has successfully completed a measurement. To further verify whether the measurement was successful, it is necessary to first analyze the data, this is done with the help of the next experiment.

6.1.3 Experiment Analyze

The purpose of this experiment is to demonstrate that the data recorded during the previous experiment is real measured data. The setup again contains one physical Sound Camera as explained in Appendix B with a pistonphone fixed on microphone number 14. The data from the previously done experiment was already uploaded to the Portal.

To verify if the experiment is successful the computed hologram at the frequency 1kHz of the pistonphone must show a clear mark at the position at which it was placed. Figure 32 shows two different holograms of the same measurement but at different frequencies. The first Hologram shows a clear mark of the 1 kHz pistonphone signal in the frequency band '892.5Hz - 1121.5Hz' at position number 14 at which it was placed. This means that both this experiment as well as the previous experiment was carried out successfully.

6.1.4 Experiment Repeatability

The purpose of this experiment is to demonstrate that the previous experiments can be repeated and again real measurement data is used and not pre-stored sample data. The experiments 'Measurement' and 'Analyze' are repeated, but now the pistonphone is placed on a different microphone. The position of the 1 kHz pistonphone during this experiment is on microphone number 19. Figure 33 shows two different holograms on the second measurement but again at
two different selected frequencies. Again the first hologram shows a clear mark at the microphone at which the pistonphone was placed. This means that the experiments can be repeated and are performed on real data.

![Hologram](image1)

(a) 892.5Hz - 1121.5Hz  
(b) 10459.5Hz - 10688.5Hz

Figure 32: The hologram from measurement #1 on the Portal hologram viewer, at two different selected frequencies bands.

![Hologram](image2)

(a) 892.5Hz - 1121.5Hz  
(b) 10459.5Hz - 10688.5Hz

Figure 33: The hologram from measurement #2 on the Portal hologram viewer, at two different selected frequencies bands.

### 6.2 Validation of Requirements

In this section all the requirements which were formulated in Section 3 will be individually be validated. The established requirements were dived in three categories: User, Software and Hardware requirements. The following subsection will corresponding to each of these categories.
and will validate the individual requirements that belong to that category. The validation will be done with the help of the decisions and the designs made during this research.

6.2.1 User Requirements

USE01 - The user can do measurements.
USE07 - The user can control the Sound Camera through the Sorama Portal.
These requirements are evaluated and validated with the use of the experiment Measurement described in Section 6.1.2

USE02 - The user can select a frequency to inspect.
USE03 - The user can inspect multiple frequencies.
These requirements are evaluated and thereby validated with the use of the experiment Measurement described in Section 6.1.3

USE04 - The user can see a list of his available Sound Cameras.
USE05 - The Sound Camera can be operated by a terminal with a web browser installed.
These requirements are evaluated and thereby validated with the use of the experiment Discovery described in Section 6.1.1

USE06 - The user cannot overspend his credit balance.
This requirement is enforced server side or in a so called creditService described in Section 4.6.3

USE08 - The user does not need an Internet connection to do measurements.
These requirements are not implemented in the prototype, but a implementation on how to implement is described in Section 5.3.4

USE09 - The Sound Camera can only be operated by the owner of the Sound Camera.
USE10 - The user need to be logged in to use the Sound Camera.
These requirements are reinforced and explained in the security perspective described in Section 4.6.4

6.2.2 Software Requirements

SOF01 - The Software Architecture is programming language independent.
The choice for SOA, as described in Section 4.2, allows the different services to be built as independent programs. This allows the programs to be built in different programming language.

SOF02 - The Architecture needs to be portable so it can support the current hardware design.
In the development perspective in Section 4.6.2 and during the choice of implementation in Section 5.1 the requirement of 'platform independence' is maintained. For example the choice for C# which is cross-platform, because of the intermediate code.

SOF03 - The designed and created software for this architecture can be updated remotely.
In Section 4.6.2 it is described how the built-in remote update functionality of Petalinux 37
is used.

SOF04 - The Sound Camera can do preprocessing of measurement data.

SOF05 - The Sound Camera support doing a single measurements for a predefined length of time.

These requirements are shown during the experiments explained in Section 6.1.

SOF06 - The Sound Camera supports doing continuous measurements for an under-mined amount of time.

SOF07 - The Sound Camera can continuously stream data to an end-point.

The current prototype does not support this, however the needs were already foreseen. Therefore a StreamingService is designed within the software architecture which is describe in Section 4.

SOF08 - Communication between the device and the online Sorama Portal is secured.

The fact that the chosen communication technique WebSocket has built-in functionality to encrypt the data on top of the socket as explained in Section 5.1.3 fulfills this requirement.

SOF09 - Communication between the linked devices is secured.

The fact that in Section 5.1.4 it states that the chosen technique ZeroMQ which is used for communication between devices. Has built-in functionality to encrypt all communication between Publishers, Subscribers and the multiple serviceBussen fulfills this requirement.

SOF10 - After the device is powered up, it makes itself known to the Sorama Portal and surrounding devices.

In Section 5.1.2 the choice of technique is explained and in Section 6.1.1 the evaluation is done with the help of an experiment which shows that the Sound Camera shows itself automatically to the Sorama Portal.

SOF11 - The device can be operated without the need of a working Internet connection.

SOF12 - The device can make off-line measurements with the use of a local storage.

Solutions for these requirements are given in the Section 5.3.4.

SOF13 - The device can respond to commands given by the online Sorama Portal.

During the experiments in Section 6.1 it is shown that the Sound Camera can be controlled remotely.

6.2.3 Hardware Requirements

HAR01 - Multiple devices can be linked together and operated as one for the costumer looking Sound Camera.

In Section 5.3.1 it is explained how multiple devices can be connected to form a compound, more powerful Sound Camera.

HAR02 - The device operates without the use of persistent storage.

By design the system is stateless. So this means that experiments in Section 6.1 could not be done of the design was not stateless.
Devices that are linked will have their internal clock synced. In Section 5.3.3 a solution is given that synchronizes the internal clocks of multiple devices.

The Architecture can run on an ARM processor.

The Sound Camera is by design stateless.

The Architecture is compatible with both versions of the Sound Camera. During the selection and evaluation of software in Section 5.1 the requirement ‘ARM compatibility’ is guarded.

The Sound Camera supports local storage. In Section 5.3.4 is explained how support can be built and used for local storage.

6.3 SWOT analysis

Table 11 shows a Strengths, Weaknesses, Opportunities & Threats (SWOT) analysis which has been done on the whole software architecture design and created prototype. The SWOT analysis is done to determine the overall Strengths, Weaknesses, Opportunities & Threats of the software architecture as a whole. Below the Table all the items are elaborated and explained in more detail.

<table>
<thead>
<tr>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 Extensibility.</td>
<td>W1 Current proof of concept is not optimal for offline possibilities.</td>
</tr>
<tr>
<td>S2 Individual service optimization</td>
<td>W2 Overhead of individual services.</td>
</tr>
<tr>
<td>S3 Isolation of individual services</td>
<td></td>
</tr>
<tr>
<td>S4 Scalability.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threats</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 C# not optimal for embedded systems.</td>
</tr>
<tr>
<td>T2 No persistent storage on platform.</td>
</tr>
<tr>
<td>T3 Overhead of individual services.</td>
</tr>
</tbody>
</table>

Table 11: SWOT analysis of the designed architecture.

Elaborations of the Strengths:

S1 - ‘Extensibility’: It is easy to extend the functionalities of the Sound Camera by creating new services or combining already made ones.

S2 - ‘Individual service optimization’: It is easy to update, extend or optimize individual services because they have well-defined interfaces and run as separate processes.

S3 - ‘Isolation of individual services’: It is possible to containerize the different services to isolate them from each other.
**S4 - Scalability**: The module hardware design of the Sound Camera allows the placement of multiple DMAIO boards. These DMAIO boards can be easily connected via the serviceBus to form a larger system.

Elaborations of the Weaknesses:

**W1 - Current proof of concept is not optimal for offline possibilities**: The current proof of concept does not have any ability to perform offline measurements.

**W2 - Overhead of individual services**: Running individual services in different processes can create overhead compared to running them in one process.

Elaborations of the Opportunities:

**O1 - Easy to access new market segments**: With the ability to easily extend the architecture with new features, it is easy to access new and different markets.

**O2 - Future possibilities to grant access to third parties**: If the services run in isolated environments it would be possible to grant third parties access to the hardware to improve usage of the Sound Camera.

Elaborations of the Threats:

**T1 - C# not optimal for embedded systems**: C# has been chosen for the proof of concept implementation besides that it is not optimal for embedded platforms compared to C++. But the most realistic option for the prototype and a real suitable option besides C++ for the real implementation, because of the already existing code base in C# of Sorama.

**T2 - No persistent storage on platform**: The hardware design of the DMAIO boards only contains non-volatile memory. This means that no user data can be permanently stored during runtime.

**T3 - Overhead of individual services**: The risk of doing the same work by multiple services, holding both when developing the services as during running the service, generates an overhead. For example two services doing the same calculation work after each other which is a waste of resources.
## Conclusion and Future work

To conclude this report the following two subsections the Conclusion and what future work can still be done are stated.

### 7.1 Conclusion

The main research question/goal was to design and develop a new software architecture for the Sorama Sound Camera. The first part of the research was orientated on defining what the current situation within Sorama is. The second part was a literature study to find out what related work was done on this field. One of the papers showed that the Sound Camera can be classified as a type 4 IoT device. The third part was defining three perspectives: User, Company and Technical perspective. These defined perspectives and the first defined terminology/situations are used to define the requirements for the Sound Camera. The fourth part was describing the software architecture using Architecture Description (AD) following the SO/IEC/IEEE 42010 standard, the Rozanski & Woods method was chosen. Rozanski & Woods describes a software architecture with the use of viewpoints and perspectives. The chosen viewpoints are: Context, Functional, Development and Deployment. The chosen perspectives are: Availability and Resilience, Development Resource, Usability and Security. The fifth part was the selection of the various techniques and the creating of a proof of concept implementation. The sixth part was validation and the evaluation of the individual requirement, the proof of concept with the help of experiments and the entire research with the help of a Strengths, Weaknesses, Opportunities & Threats (SWOT) analyze.

The research showed that a Service Oriented Architecture (SOA) pattern is the best choice as the base for the new software architecture. This approach defines the functionalities of the Sound Camera as a combination of services. This allows new functionalities of the Sound Camera to be added by creating new services, or composing existing services in a different manner. Also the possibility for services to be started or stopped on-the-fly, makes it possible to change the functionality of the Sound Camera in an instant. Making it an optimal flexible software architecture for the Sorama Sound Camera.

During the selection of the various technologies used for building the first services for the software architecture and which are used in the proof of concept, security was an important part of the considerations. The chance that a Sound Camera is used in a R&D department of a customer is very likely. The choice between a technology with or without encryption integrated, the technique with encryption was chosen. The selection showed that for all needed technologies a candidate with encryption was available and was chosen. These chosen techniques ensures that all the internal and external communication are secured. Therefore the Sound Camera is safe for deployment and usage within R&D departments and others areas which works with high sensitive data.

To demonstrate the working and practicality of the new software architecture, a proof of concept was made. The created proof of concept is able to do measurements with real microphone data and is able to send this data to the Sorama Portal to be analyzed. To present a hologram of the measurement a C# FFT implementation is created. This also shows that it is possible to do preprocessing on the Sound Camera. During the making of the Proof of concept the Avahi Zeroconf discovery programming was compiled and installed on the DMAIO board. This Avahi Zeroconf implementation effort is now already used by the internal development team to use as the default discovery method of their implemented software architecture running on the dmaio board. Besides Avahi the complete tool-chain and compiler packaged Mono was compiled and installed on the dmaio board. Mono allows the usage of the programming language C# for...
the creation of the needed services. With the given instructions, it is possible for everybody to compile *Avahi* or *Mono* for every *Petalinux* deployment.

The final conclusion is that the designed software architecture is not designed as a single functionality application. It is designed as a process to go from point A to point B, which can be adapted to the needs of the situation in which the Sound Camera is used. For this research it was restricted to product development by Sorama’s customers. However if the Sound Camera would be used for example in smart city’s, only new services needs to be added or rearranged to fulfill this situation and not a complete redesign of the software architecture.

### 7.2 Future work

For every research it is possible to do some future work, that also holds for this research.

No effort is put into how Sorama credits can be handled safely by the software architecture. A current development are Blockchain systems like for example Bitcoins. Blockchain allows Sorama credits to be remotely stored in offline situation without the possibility to tamper with, because validation up to the chain on which it was originated is possible. Therefor the Blockchain technique would be worth to further investigate and how it can be used for distribution Sorama Credits.

The current research and proof of concept did not focus on how to on-the-fly change the functionalities of the Sound Camera. More research is needed to for example build a service repository in the software architecture which can be used to find and start new or different services on the Sound Camera.

Some effort is put into finding out if it is possible to separate each service into virtual independent environments by virtual containers or virtual machines to have more isolation, control of the resources in terms of memory/performance usage and fault tolerance of the whole system. Then a failure of an individual service will not affect the total system. In Section 5.1.6 a small selection is done and it is by this research advices to use *LXC* as the base for making containers in Linux.

Only a suggestion is given on how to make offline measurements possible by building a storage service on the users PC and offer this via Zeroconf to the Sound Camera. The architecture allow the creation of a service which could serve a stripped down version of the Sorama Portal locally to the user. With this small Portal, the user can do simple tasks like doing measurements and validating the result. Local storage can be offered by inserting an USB thumbdrive in to the Sound Camera.

To be more confidence about the deployability within R&D and other high sensitive environments, it is needed to re-ensure costumers of the strength of the chosen security. More effort is required in verification and certification of the used security protocols. Not only on a level of implementation but also on the level of development. For example how are the used private and public certificates stored within the development tool-chain.
References


A Class Diagram
B Proof of concept Setup

This appendix will explain the experimental setup used during the testing of the proof of concept implementation. During the experiments a pistonphone is used as a sound source, Table 12 shows the specification of the pistonphone. The pistonphone was fixed in position with a lab stand. By using a small cap on the pistonphone, the sound generated could be focused on only one microphone. A photo of the setup used during the experiments can be seen in Figure 34. Figure 35 shows the mapping of the microphones on the Sound Camera used during the experiments. The mapping is used to number the microphones, thereby a number can be presented and then it is clear to what microphone it belongs on the Sound Camera.

No special measures have been taken to determine or control the environment in which the measurements are been carried out.

For the proof of concept implementation a small mock-up Portal has been developed, the specification of this mock-up portal can be found in Table 13. Figure 36 shows a screenshot of the created mock-up web Portal. The user can see his available Sound Camera, his made measurements and a console in which all the communication between the Portal and the Sound Cameras can be viewed.

<table>
<thead>
<tr>
<th>Brand</th>
<th>CESVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>CB006</td>
</tr>
<tr>
<td>Output</td>
<td>94dB@1kHz</td>
</tr>
<tr>
<td>Frequency</td>
<td>1 kHz</td>
</tr>
<tr>
<td>Reference No</td>
<td>PTB-1.61-4027729</td>
</tr>
</tbody>
</table>

Table 12: Specification of the pistonphone used during the experiments

Figure 34: Photo of pistonphone fixed on lab stand for a cam64.
Figure 35: Microphone mapping of Cam64 used during experiments.

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>PHP &amp; HTML5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>Zend Framework 2</td>
</tr>
<tr>
<td>Database</td>
<td>MySQL</td>
</tr>
<tr>
<td>Webserver</td>
<td>Apache</td>
</tr>
<tr>
<td>Websocket handler</td>
<td>NodeJS</td>
</tr>
</tbody>
</table>

Table 13: Specification of the mockup Portal

Figure 36: Screenshot of the mockup Portal, which was made for the proof of concept implementation.
C Installation of Packages

In this appendix the installation of the following programs on the DMAIO is elaborated:

Mono: Is used to supply the CLR for C#.

ZeroMQ: is used as the serviceBus in the designed software architecture.

Avahi: is used for Zeroconf discovery in the proof of concept.

The following outline is used to explain the installation:

Introduction: A small introduction of the package.

Usage: For what it is used within the architecture or proof of concept.

Information: Displays the version information.

Dependencies: What are the dependencies of the packages?

Installation: Which commands needs to be exuded and which Flags are used.

Command Explanations: Explains all the used Flags of the installation section.

C.1 Installation of Mono

Introduction

Mono is an open source implementation of the Microsoft .NET framework, the language C# and the Common Language Runtime (CLR). This allows users to use mono for cross platform application, and makes C# possible on for example Linux.

Usage

Mono is used for the to make it possible to use the Microsoft .NET framework and C# on the Sound Camera.

Information

version: 4.0.1

Dependencies

• glib-2.0

Installation

```bash
./configure --prefix= CC=/opt/PetaLinux/petalinux-v2014.4-final/tools/linux-i386/arm-xilinx-linux-gnueabi/bin/arm-xilinx-linux-gnueabi-/gcc --host=arm-linux
make
make DESTDIR="/home/tijl/mono/build/mono"
```

Command Explanations

CC: C++ cross compiler of Petalinux

–prefix: The prefix which will be added to all location of files

–host: The target platform for which we compile mono

DESTDIR: The root destination at which the compiled files will be copied

Folder structure

The folder structure of mono:

• bin
  – mono

• lib
  – mono
    * 2.0
    * 3.5
    * 4.0
    * 4.5
    * gac

• etc
  – mono
    * 2.0
    * 4.0
    * 4.5

C.2 Installation of ZeroMQ

Introduction

ZeroMQ also known as ØMQ is a distributed messages engine. ZeroMQ describes themself as "A ØMQ socket is what you get when you take a normal TCP socket, inject it with a mix of radioactive isotopes stolen from a secret Soviet atomic research project, bombard it with 1950-era cosmic rays, and put it into the hands of a drug-addled comic book author with a badly-disguised fetish for bulging muscles clad in spandex."

Usage

ZeroMQ is used for the serviceBus internally and externally communication on the Sound Camera.

Information

version: 4.1.3

Dependencies

• libsodium-1.0.3
Installation

```
./configure --prefix="" CC="/opt/PetaLinux/petalinux-v2014.4-final/tools/linux-i386/arm-xilinx-linux-gnueabi/bin/arm-xilinx-linux-gnueabi-gcc" CXX="/opt/PetaLinux/petalinux-v2014.4-final/tools/linux-i386/arm-xilinx-linux-gnueabi/bin/arm-xilinx-linux-gnueabi-c++" --host=arm-linux sodium_CFLAGS="-I/home/tijl/zeromq/build/libsodium/include" sodium_LIBS="-L/home/tijl/zeromq/build/libso/libso/libso/libso/libsodium"
```

make

make DESTDIR="/home/tijl/zeromq/build/zeromq"

Command Explanations

- `CC`: C++ cross compiler of petalinux
- `CXX`: C cross compiler of petalinux
- `--prefix`: The prefix which will be added to all location of files
- `--host`: The target platform for which we compile mono
- `sodium_CFLAGS`: The flags for dependency libsodium
- `DESTDIR`: The root destination at which the compiled files will be copied

C.3 Installation of Avahi

Introduction

Avahi is a system which facilitate device and service discovery via mDNS and DNS-SD protocol. This gives the user the possibility to find the devices and it services without any configuration. (ZeroConf) Avahi is a free implementation of its better known Apple Bonjour version.

Usage

Avahi is used for the internal device and service discovery of the Sound Camera.

Information

version: 0.6.31

Dependencies

- dbus-1.10.0
- expat-2.1.0
- libdaemon-0.14
Installation

```bash
PKG_CONFIG_PATH="/home/tijl/avahi/build/dbus/lib/pkgconfig:/home/tijl/avahi/build/libdaemon/lib/pkgconfig"

make

make DESTDIR="/home/tijl/avahi/build/avahi"
```

Command Explanations

- **CC**: C++ cross compiler of Petalinux.
- **--prefix**: The prefix which will be added to all location of files.
- **--host**: The target platform for which we compile mono.
- **--with-distro**: Disable the bootscript.
- **--disable-nls**: This parameter disables the use of NLS.
- **--disable-glib**: This parameter disables the use of Glib.
- **--disable-gobject**: This parameter disables the use of GObject.
- **--disable-gdbm**: This parameter disables the use of GDBM.
- **--disable-qt3**: This parameter disables the use of QT3.
- **--disable-qt4**: This parameter disables the use of QT4.
- **--disable-gtk**: This parameter disables the use of GTK.
- **--disable-gtk3**: This parameter disables the use of GTK3.
- **--disable-mono**: This parameter disables the use of Mono.
- **--disable-monodoc**: This parameter disables the generation of MonoDoc.
- **--disable-python**: This parameter disables the use of Python.
- **--disable-doxygen-do**: This parameter disables the generation of doxygen.
- **--disable-manpages**: This parameter disables the generation of documentation.
- **--with-autoipd-user=avahi**: This parameter sets the user under which the process will run.
- **--with-autoipd-group=avahi**: This parameter sets the group under which the process will run.

**PKG_CONFIG_PATH**: This parameter sets the location of the PKG Config.

**DESTDIR**: The root destination at which the compiled files will be copied.