A Fog Architecture to Support Intramural Care Based on Multiple Types of Wearables

Abstract

This technical report proposes a design solution of a system architecture for wearables in the context of intra-mural use case. A fog architecture approach, in which the measurement data of multiple different types of wearables can be collected, if needed transformed and processed and sent to a software platform, was used. Moreover, the given design solution is flexible making it applicable for other similar use cases. An available work analysis is presented surveying already existing software architectures for wearables and a software architecture pattern, originally created for mobile sensing, to make the software architecture flexible. A technical design was created via an iterative process. The most essential parts of the design were implemented in a proof of concept and a Scenario-based analysis of software architecture was done to verify the design. Finally, the outcomes of the analysis and proof of concept which includes chosen techniques and a possible bottleneck, are discussed.

1. Introduction

The research project *All-round Gateway for IoT Signals* (AGIS) was part of a bigger research project called *Streaming Wearable Stress measurement Platform* (SWSP). The SWSP project focusses on investigating the usefulness of wearables within health care to measure stress. Technical, social and legal aspects are taken into consideration and the ultimate goal is to deliver a wearable solution to health care authorities.

Within the SWSP project EPICs were defined. To get a clear description of the current focus (patients with dementia living in closed nursing homes) of the SWSP project, a use case was defined based on the, to the current focus related, EPIC and technical requirements which were defined in a previous sub-project of the SWSP project. More information about how this use case definition was drafted can be found in appendix A. The defined use case, which will be the main use case for the AGIS project, is as follows:

“The system should support the clinical reasoning of healthcare professionals to optimise the quality of living of clients. Clients are persons with dementia and misunderstood behaviour. Personal stress levels of clients are measured by biometric values using wearables. The clients live in a closed nursing home and are not allowed to go outside. The nursing home can be multiple floors. The wearables should work around the entire building. All the wearable data should be collected, and if need, transformed and processed so it can be sent to a software back-end. During a crisis situation a patient needs to be measured and the measurement data needs to be made visible per 1 second. During a non-crisis situation this interval will be more than 1 second. This interval also depends on what the wearable supports. Within the nursing home different types of wearables can be used. The data will be made visible in a front-end and can be used by healthcare professionals to analyse the data afterwards to take substantiated decisions to improve the quality of living of clients.”

As described in the use case definition it should be possible to use multiple different types of wearables. Currently, using multiple different types of wearables creates a potential problem of measurement data being stored and shown on different places caused by the wearables all having their own apps/dashboards. A common approach with third party wearables is to use a phone as gateway to show the measurement data and/or sent it to a cloud e.g., the MoveSense HR2 [1] and the EmbracePlus [2]. The product owner of the SWSP project made clear that using a phone for patients with dementia is not convenient because the patients might not always carry their phone with them and cannot be made conscious that they need to always carry their phone to let the wearables work. Another difference between the desired solution described in the use case and the, currently by third party available wearable, solutions is the person who needs to gain insights on the wearable measurement data. The use case requires health care professionals to gain insights on measurement data of their clients. Those three potential problems create the need for a solution which resolves the problems and enables the use of third-party wearables in the, as in the use case described, desired way.

The AGIS project has two main research goals. The first goal is to propose and evaluate a technical design that can be implemented to create a system that can gather information from one or more different wearables on a gateway like device, if needed transform and process this data and send it to a software back-end. Figure 1 shows a context diagram of the desired system. The technical design is going to be created for the device which is indicated by the red box. The second goal is to create the technical design in a way so that it is also applicable for other similar use cases. This requires the design to be flexible.

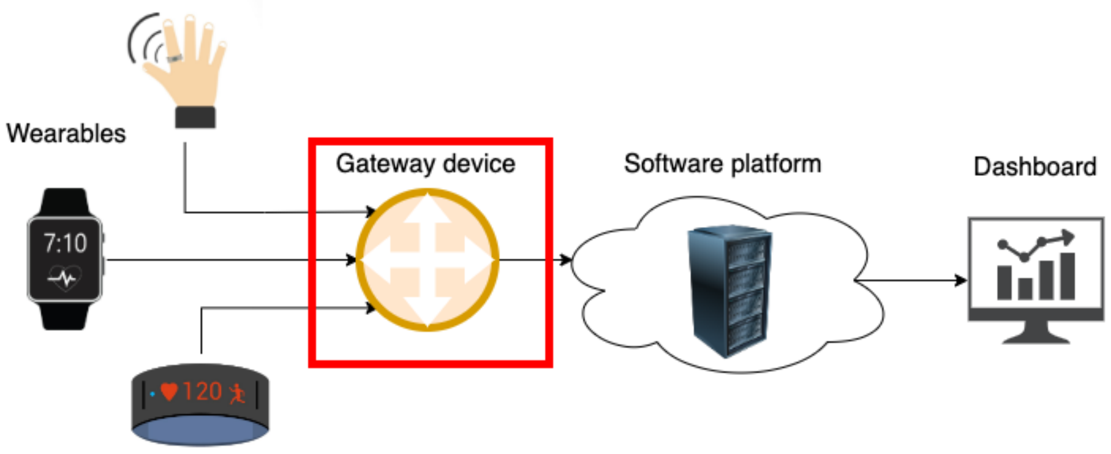

Figure 1 Context diagram

The project has links to both the system engineering and the software engineering domains. To avoid confusion about terms that have different meanings in the two domains a project dictionary is shown in Table 1. The dictionary was drafted together with another researcher within the SWSP project who, being a software engineer, was responsible for engineering the software platform in the cloud [3]. The dictionary contains terms which are used to describe the system, the design, and the functionality.

Term	Description
Wearable	A device a human can wear which measures biometrics
Gateway device	A device in between the wearable and the software platform
API Gateway	A central hub that manages and optimizes communication between clients and multiple microservices.
Software platform	Microservices with their endpoints, Storage, AI, other logic
Cloud	Azure environment
Fog device	The device between the wearable and the software platform which is designed with the fog approach meaning that it will have the possibility to not only forward the data from the wearable to the software platform but also in between transform and/or process the data.
Node	A single stand-alone software application (Comparable to 1 microservice)
Node group	A combination of nodes from the same type. For example, a transform node group is a combination of a GSR_Transform_Node which transforms the GSR measurement data from Arduino counts to micro siemens and a TimeStamp_Transform_Node which transforms Unix time to a yyyy/mm/dd format.
Load balancing internal	Dividing the tasks between different identical nodes to spread the workload.
Load balancing external	Dividing the tasks between different Fog devices to spread the workload.
Stress management System	The combination of all separate software applications and devices. In project context the system will be: The wearables + the Fog devices with their nodes + the software platform
Gateway system	The hardware and software of the gateway device.
System	A set of hardware and software working together to serve a defined purpose.

Table 1 Project dictionary

The rest of this report is organized as follows. Section 2 explains the design pattern which is used for the project. It explains in detail what the process of creating and verifying the design looked like and explains the methods used and the reasoning and added value of creating a proof of concept. Section 3 shows an available work analysis in which existing software architectures in wearable systems are researched. This section also shows a system architecture pattern which can be used to make the system flexible. Section 4 shows the results. This section shows diagrams of the final design with rationales of the main discissions. It also gives technical information about the proof of concept which has been made. Section 5 discusses the different iterations of the design, findings about the used techniques for the proof of concept and other interesting findings. The discussion section ends with possibly interesting future work. Section 6 shows the threats to validity and section 7 shows the conclusion of the technical report.

2. Method

2.1. Realise as an expert research pattern

The main overall method used for this project is the "Realise as an expert" research pattern of the DOT-framework [4]. This pattern consists of three stages. During the first stage useful techniques and/or solutions are investigated. During the second stage a new solution is created with the help of the techniques and solutions found during the first stage. The final stage of the design pattern is verifying if the solution works correctly. The subsections below will further elaborate on the three stages and how they were executed during this research.

2.1.1. First stage

To gather useful techniques and solutions an available work analysis [5], which is described in section 3, was done. This available work analysis resulted in finding three different architecture approaches and a software architecture pattern to create flexibility within the design.

2.1.2. Second stage

The goal of the second stage is to come up with a solution using the results of the available work analysis. During this stage the IT architecture sketching [6] and brainstorm method [7] were used to come up with a design. Both, co-researchers of the project and an external expert were used to discuss and brainstorm about the design.

2.1.3. Third stage

The third stage consists of two substages. The first substage is creating a proof of concept, using the prototyping method [8], to verify that the software architecture is suited for the main use case. The second substage is using the scenario-based analysis of software architecture method [9] to verify if the software architecture is suited for similar use cases. Both approaches are described in the subsections below.

2.1.3.1. Proof of concept

Multiple iterations of a proof of concept were used to verify the design. To make sure the verification of the design was carefully done and implementing it in a proof of concept was manageable the designed gateway system was divided into seven main functionalities: Nodes communication (a), Wearable data extraction (b), Dynamic routing (c), load balancing between nodes (Internal load balancing) (d), load balancing between devices (External load balancing) (e), Scalability (f) and Software platform communication (g). These seven main functionalities were drafted based on the main use case description. Table 2 shows the seven main functionalities and how they correspond to parts of the main use case definition.

Main Functionality	**Main use case definition**	**Explanation**
(a) Nodes communication	n/a	This functionality was not drafted based on the use case. This is the foundation to being able to implement any of the functionalities.
(b) Wearable data extraction	"All the wearable data should be collected" "Within the nursing home different types of wearables can be used"	Wearable data should be collected. Different types of wearables might result in different ways of data collection.
(c) Dynamic routing	"Within the nursing home different types of wearables can be used"	Different types of wearables can possibly mean different type of measurement data. Different type of measurement data can require different action to process the measurement data. Therefore, dynamic routing is required.
(d) Load balancing between nodes	"During a crisis situation a patient needs to be measured and the measurement data needs to be made visible per 1 second. During a non-crisis situation this interval will be more than 1 second." "The clients live in a closed nursing home and are not allowed to go outside." "The wearables should work around the entire building"	The required measurement interval depends on the situation. It is important that the architecture can divide the load between nodes so that both low and high data intervals can be handled. In a closed nursing home, the number of clients can differ. More clients means more wearables thus more wearable measurement data. The system should be able to handle both a low and high amount of data. Load balancing between nodes is required to help with handling high amounts of data. The wearables should work around the entire building. If the building contains multiple gateway devices but patients are all in the same room (E.g., during dinner) the required load of a single gateway device needs to be increased. Load balancing between nodes is required.
(e) Load balancing between devices	"During a crisis situation patient measured data needs to be measured and made visible per 1 second. During	The required measurement interval depends on the situation. It is important that the architecture can divide the load between devices, when dividing load between nodes does not solve the problem anymore,

	a non-crisis situation this interval will be more than 1 second.” “The clients live in a closed nursing home and are not allowed to go outside.” “The wearables should work around the entire building”	so that both low and high data intervals can be handled. In a closed nursing home, the number of clients can differ. More clients means more wearables thus more wearable measurement data. The system should be able to handle both a low and high amount of data. Load balancing between devices, when load balancing between nodes does not solve the problem anymore, is required to help with handling high amounts of data. The wearables should work around the entire building. If the building contains multiple gateway devices but patients are all in the same room (E.g., during dinner) the required load of a single gateway device needs to be increased. If load balancing between nodes does not help anymore, load balancing between devices is required.
(f) Scalability	“During a crisis situation patient measured data needs to be measured and made visible per 1 second. During a non-crisis situation this interval will be more than 1 second.” “The clients live in a closed nursing home and are not allowed to go outside.” “The wearables should work around the entire building”	During a high measurement interval the system should be able to scale. Scaling can be done internally and works both ways. When the load is high it should scale up and when the load is low it should scale down. In a closed nursing home, the number of clients can differ. More clients means more wearables thus more wearable measurement data. The solution should be able to scale both internally and externally. To be able to retrieve wearable data at only location within the building the system physically needs to be able to scale. This means that it should be possible to add gateway devices
(g) software platform communication	“sent to a software back-end”	The gateway device should be able to sent its data to a software back-end.

Table 2 Main functionalities with corresponding main use case definition parts

The iterations of the proof of concept builds on the code of the previous iteration. Due to the limited time and the challenge of verifying abstract quality attributes like modifiability or security which provide very little procedural support for evaluating them [9] the goal of the proof of concept was not to verify whether the proposed design for the functionalities is the “best” option. The goal of the proof of concept was to implement essential parts of the gateway system to verify if the design is realisable, if there are no mistakes in the design and if there are no missing parts. Because the goal of the proof of concept was verifying the architecture the chosen techniques within the proof of concept are purely chosen on the basis of ease of implementation. The discussion section will elaborate more on the chosen techniques and shed light on whether the chosen techniques are recommended for an actual implementation of the solution or not.

2.1.3.2. Scenario-based analysis of software architecture

The software architecture was verified by using the Scenario-based analysis of software architecture method [9]. This method was recommended by a senior researcher of the SWSP project and proposes a way of verifying software architecture using the SAAM (Software architecture Analysis Method) [9].

3. Available work analysis

An available work analysis was done. The available work analysis is the first stage of the realise as an expert research pattern as described in subsection 2.1.1. Google scholar was used to find literature related to software architectures of wearable systems. The following main search query was used: “Software architecture of wearable system”. A more detailed description about the available work analysis can be found in appendix B.

3.1. Software architecture

In almost every publication that was analysed the so-called edge, fog and cloud architectures were mentioned. Figure 2 shows an overview of the three different architecture layers. Cloud architecture is a form where data is stored and or processed online on server(s) and can be accessed from any device [10] [11]. Edge architecture brings data processing closer to the edge of the network. Edge devices are the devices who measure the data e.g., a Fitbit. Sometimes data processing can be executed on the measurement device itself, sometimes it is a separate device really close to the measurement device [11] [12]. Fog architecture adds a layer in between edge and cloud. The fog architecture layer can receive data from edge devices and (partly) process this data. It can also distinguish data which is relevant to send to the cloud. Fog architectures can consist of multiple devices thus making it a decentralised system. Each fog node can communicate with each other and workload can be divided over different fog nodes [11] [12] [13].

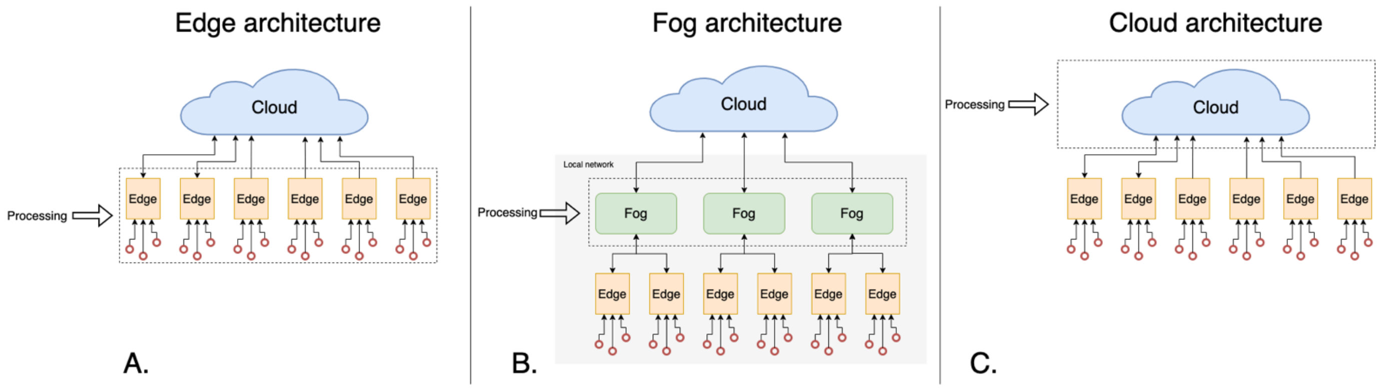

Figure 2 Edge (A), Fog (B), Cloud (C) architecture diagram

The cloud architecture has high computation power and storage capacity and can therefore handle performance heavy AI [14]. On the other hand, using a cloud architecture can increase the risk of network problems and latency and also increase privacy and security risks. This is all due to the fact that the measured information needs to be sent over the internet to the cloud. A fog architecture will process the measurement information within the same local network as the wearables. This decreases network problems, latency and privacy and security risks [14] [15] [16]. A potential bottleneck of a fog architecture is the limited computational power on a single fog node [14]. The standard fog node architecture does support the option of having multiple fog nodes and load balancing which might mitigate this bottleneck. The properties of an edge architecture are similar to those of a fog architecture but are more extreme. The computational power of a (wearable) edge device is even lower compared to the computational power of a fog device. In contrast, the security and privacy risks and the network usage and problems are less compared to a fog node architecture [14].

The use case which is being focused on for this project has a need of real time monitoring thus making latency an important factor. The use case requires to measure biometrics of clients/patients in a health care environment. This makes privacy and security also in important factor. Reduced latency and privacy and security risks were a recurring theme in almost all the literature as described in the previous paragraph. Therefore, for this project there is chosen to make use of a fog architecture and thus will be the focus of this research.

3.2. Software architecture pattern

The goal of this project is not only to design a gateway system which is suitable for the defined use case but it also suitable for similar other use cases e.g., a use case in which multiple wearables are used for a preventive screening of the falling risk and mobility of elderly in a neighbourhood. A publication was found in which two software architecture patterns named, “Sampling Package” and “Data Transformer”, were proposed which possibly could help to make the software architecture design flexible [17]. Both software architecture patterns were originally created for mobile sensing.

The first software architecture pattern is sampling package. Sample packages are separate modules within a system where each module is responsible for extracting data of a specific sensor or device. This means that a sampling package specifies what data is going to be collected, the format of the data and how the data is extracted from the sensor or device.

The second software architecture pattern is Data transformer. Data transformers are responsible for transforming the, by the sampling package extracted data. An example of this is a privacy transformer.

In the privacy transformer, privacy sensitive data can be encrypted. Each single data transformation has its own package.

Those software architecture patterns look promising. The modularity of the architecture patterns causes packages to have single responsibility thus not relying on other functional parts of the system which should make it easy to add, edit or remove packages [18] when the solution is used in other similar use cases. Because each wearable would have its own sampling package connecting and extracting data from different types of wearables should not cause big implementation problems. The modularity also makes it easy to scale because if the load on a certain package becomes really high a new identical package can be launch and the load can be divided. Since the main use case requires the option of applying algorithm or AI on the measurement data a processer layer could be added to the sampling and transform layers using the same separate nodes and single responsibility approach. During the design phase of the project, this approach will be used to find out if it is indeed applicable to this project. In the rest of the technical report, sample packages, transformers and processors will be called sampling nodes, transform nodes and process nodes. for consistency.

4. Results

4.1. Design

The second stage of the realise as an expert research pattern, as described in subsection 2.1.2., is coming up with a solution using the results of the available work analysis. The subsections below show the proposed design.

4.1.1. Architecture overview

Figure 3 shows general overview of the architecture. The architecture consists of three types of entities. The first type of entity are the wearables. These are third party devices that measure biometrics and send their measured data via a communication protocol like Bluetooth Low Energy (BLE). The data is sent to a Fog device, which is the second type of entity. This entity is a device which is responsible for extracting the data from the wearables, if needed transform and process it and send it to a software platform. It is possible that the gateway system contains multiple Fog devices. This enables the possibility of the workload to be spread over the different devices causing the system to have a higher capacity and cover a bigger area in terms of communication range. The third type of entity is the software platform. For this project the development of the software platform is out of scope. It was realised by a co-researcher within the SWSP project [19].

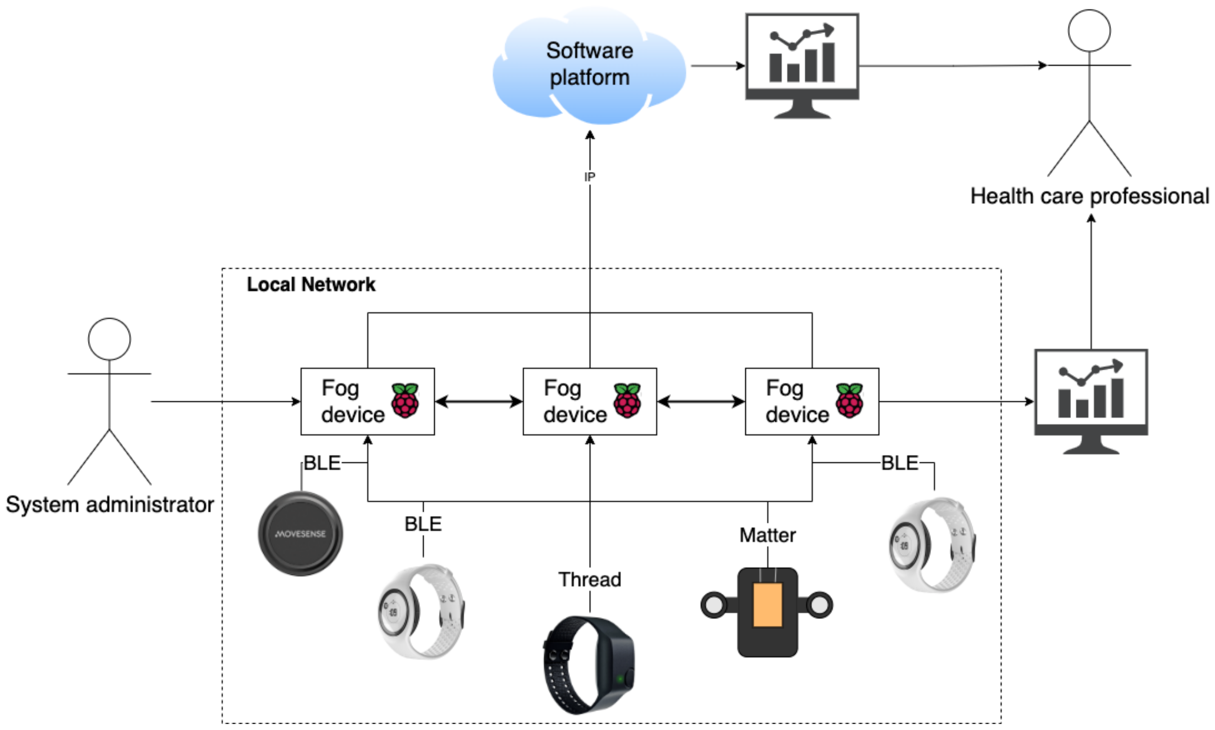

Figure 3 Architecture overview

4.1.2. Object diagram

Figure 4 shows the objects of the software architecture. The three dotted boxes (Device layer, Fog device and software platform layer) can be mapped on the three entity types of the general architecture overview of Figure 3. In this diagram the design of the Fog device is shown in more detail. The process within the fog device can be divided into four steps: Sampling, Transforming, Processing and Communication. The Sampling step is responsible for gathering wearable data. The Transforming step is responsible for transforming the data. An example of this is transforming a device specific timestamp into a universal time format. The process step is responsible for processing the data. In this step algorithms and AI can be applied to the data. The communication step is responsible for communicating the data to the user via a software platform or directly, via a web interface. The Transform and Process steps are optional since not every type of measurement needs to be transformed or processed.

The Fog device consists of four type of node groups which are equal to the four steps. A node group is a group of nodes from the same step but with different behaviour. For example, the transform node group contains a node for transforming the timestamp and a node for encrypting privacy sensitive fields. Both nodes are transforming the data and therefore belong to the transform node group, but their behaviour is different. Each node group also contains a load balancer node. If a specific node cannot handle the load a new identical node can be launched so the workload can be spread. Both the launching and spreading of the workload is done by the internal load balancer. The nodes in the four node groups drawn in this diagram are just examples. Depending on the use case different nodes might be needed.

Apart from the four step-related node groups there is also a control node group. This node group contains three types of nodes which can have multiple instances if the workload requires so. The first node is the routing node. This node is responsible for routing the measured data through the four data related node groups. The routing is dynamic so each sensor can have its own route. The second node in the control node group is the external load balancer node. Load balancing can be done both internally and externally. When the fog device still has enough computational power, but the current nodes just take too much time to do their job internal load balancing will be executed by the internal load balancer nodes. When the computational power of the fog devices is reaching its limit internal load balancing is not possible anymore. In this case external load balancing will be applied. The gateway system architecture can contain multiple fog devices as shown on Figure 3. Therefore, with external load balancing the measured data will be sent to another fog device.

The last node is the log/configuration node which is paired with a web interface node. This node logs the events of the routing and via this node configuration changes to the gateway system can be made. Via the attached web interface node, a system administrator can read the logs and configure the system.

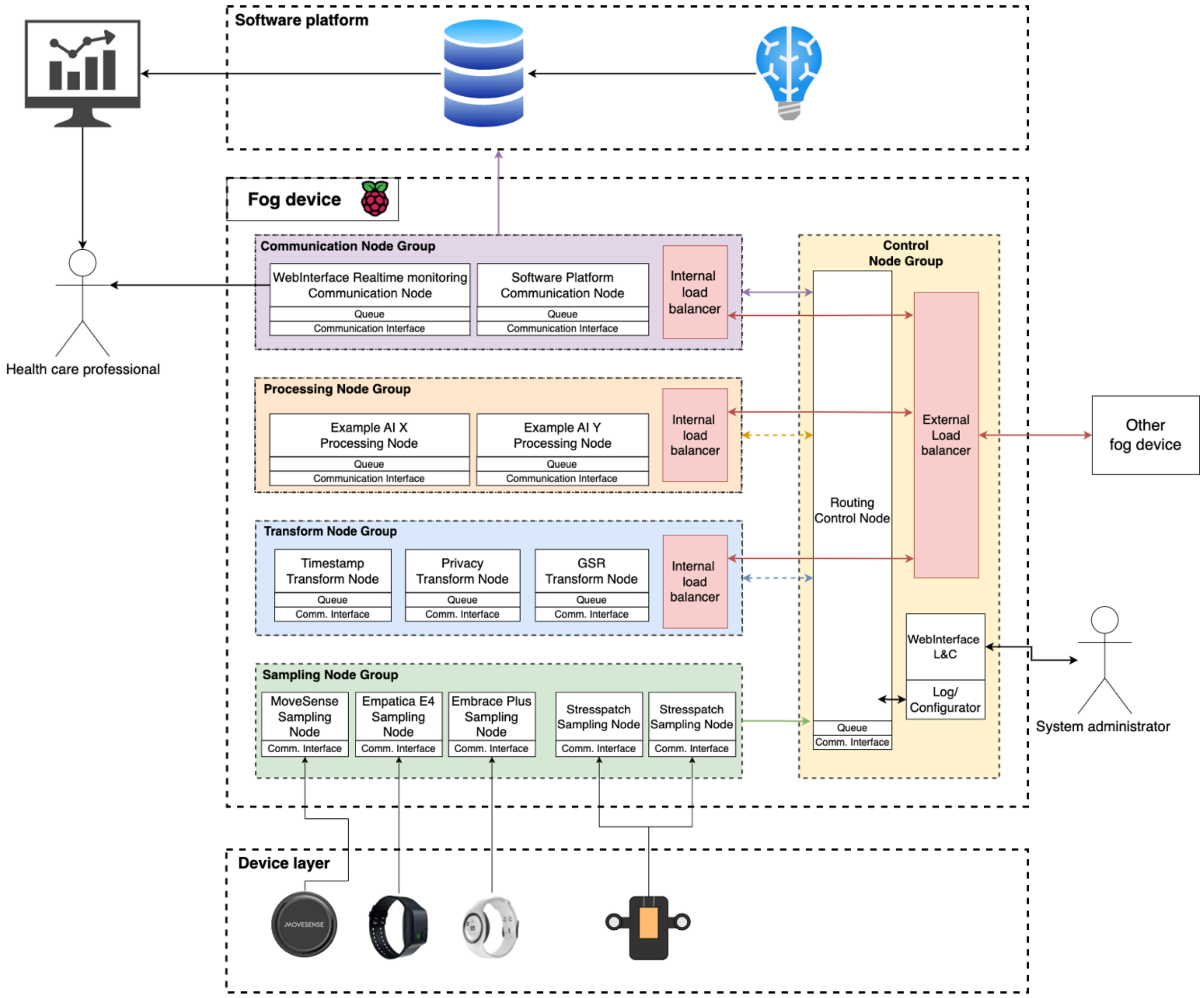

Figure 4 Object diagram

4.1.3. Gateway system flow

Figure 4 shows a static diagram of the software architecture. In this diagram the flow of the gateway system is not shown. To verify that the required flow of the gateway system works with the proposed software architecture, sequence diagrams and flowcharts have been made. The nodes within each node group differ depending on the use case. As a result, there are many possibilities of the flow of the gateway system. During this project sequence diagrams for the essential parts of the gateway system were drawn with the dementia use case as target. Figure 5 shows one of the sequence diagrams of a regular flow in which there are two different nodes in the transform node group. The rest of the sequence diagrams and flowcharts can be found in appendix C.

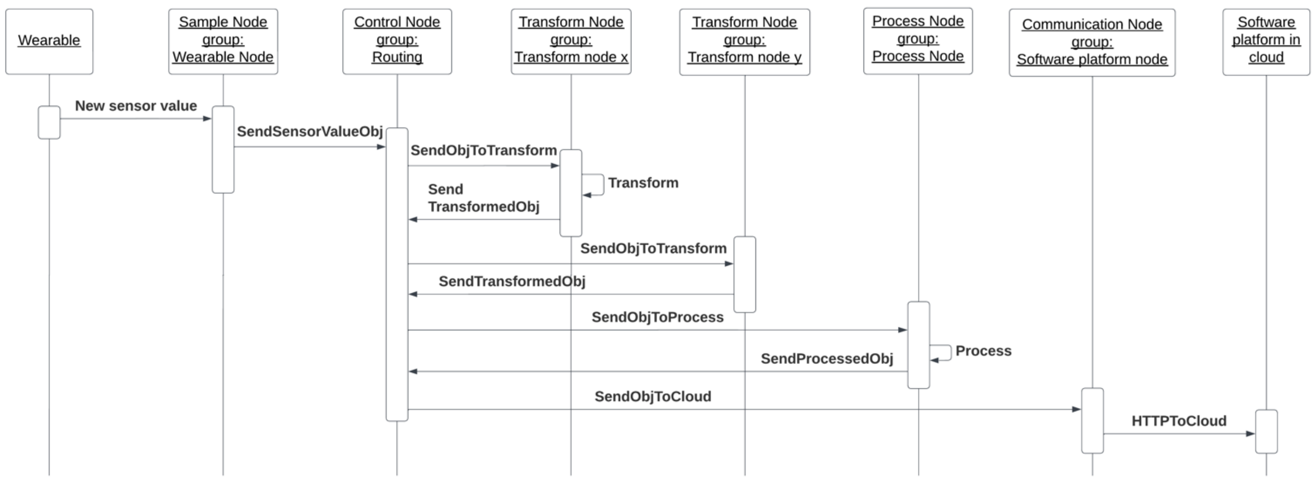
Wearable
Sample Node group: Wearable Node
Control Node group: Routing
Transform Node group: Transform node x
Transform Node group: Transform node y
Process Node group: Process Node
Communication Node group: Software platform node
Software platform in cloud
New sensor value
SendSensorValueObj
SendObjToTransform
Transform
Send TransformedObj
SendObjToTransform
SendTransformedObj
SendObjToProcess
Process
SendProcessedObj
SendObjToCloud
HTTPToCloud

Figure 5 Sequence diagram regular flow with two transform nodes

4.2. Proof of concept

Figure 6 shows all the nodes which the proof of concept of the gateway system contains. The proof of concept is the first substage of the third stage, as described in subsection 2.1.3.1. The proof of concept runs on a Raspberry Pi with Raspberry Pi OS installed running a combination of C++ and python nodes. The sample node group consists of three nodes, where two of them are written in Python which retrieve data from a MoveSense HR2 and StressPatch wearable and one is written in C++ and simulates a wearable. The proof of concept further consists of a routing node, a transform timestamp node, a processing node and a software platform node. The transform timestamp, processing and software platform node are just for simulation and do not have an actual functional implementation besides receiving and sending the message back. The transform node group contains an internal load balancer node which can divide the load between multiple transform timestamp nodes. The nodes use TCP sockets to communicate with each other.

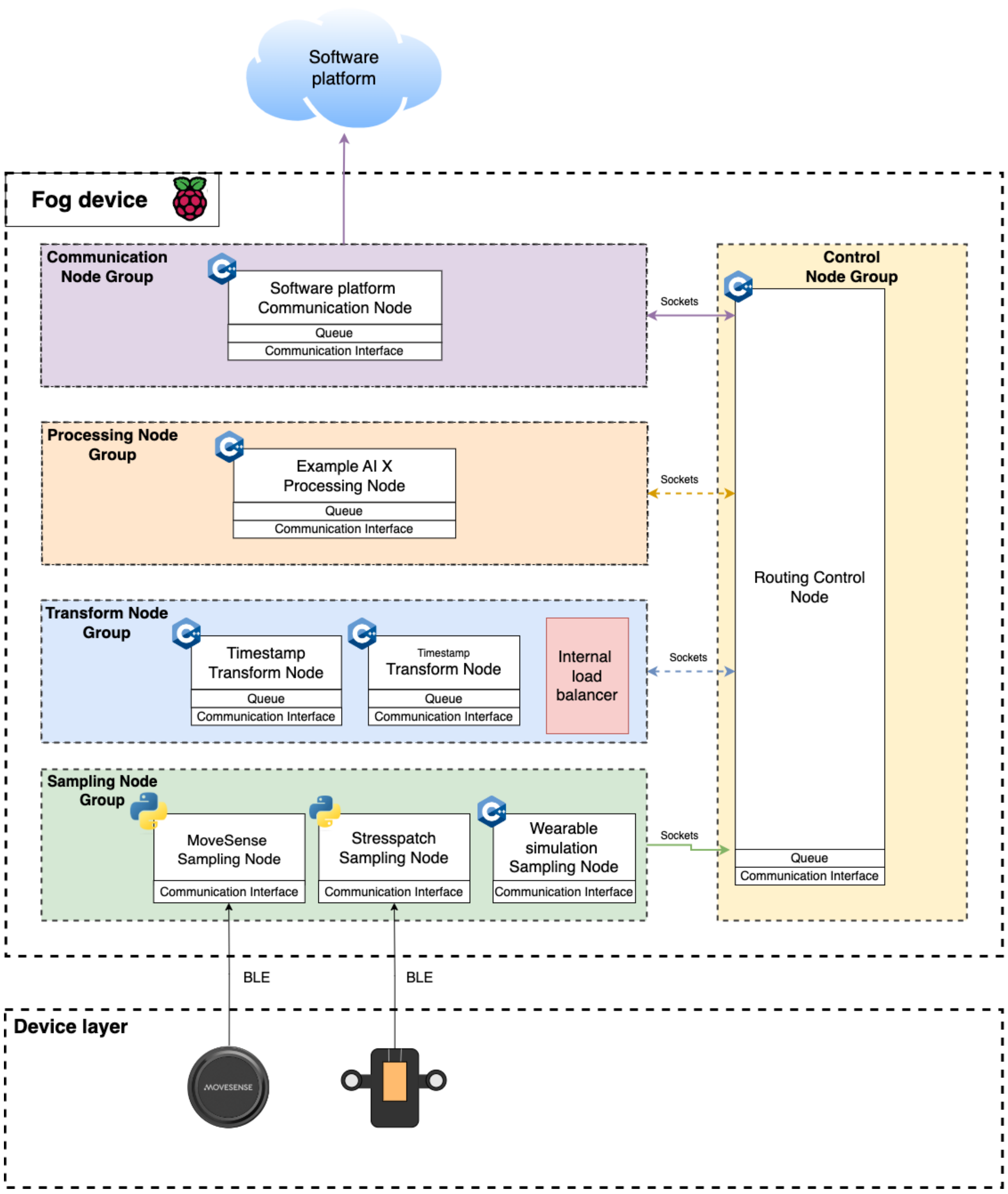

Figure 6 Object diagram Proof of Concept

4.3. Scenario-based analysis of software architecture

The scenario-based analysis is the second substage of the third stage, as described in subsection 2.1.3.2. The goal of doing a scenario-based analysis of the software architecture within this project is to verify how flexible the proposed software architecture is to use in other similar use cases [9]. Therefore, the different scenarios will be different use cases instead of specific functionalities of the gateway system. The scenarios in this analysis are drafted by brainstorming [7]. Due to limited time the focus of the brainstorm session was mainly to come up with scenarios in which changes are made to the base functionality of the main use case and possible changes at vulnerable points of the design. The base functionalities were drafted based on the main use case and are defined as, wearable data extraction, data transformation and processing and software platform communication. Table 3 shows the drafted scenarios and which changes are needed.

#	Scenario Description	Required changes
1	A use case in which the measurement data of the wearable is different than Galvanic Skin Response, Heartrate or heatflux.	No changes
2	A use case in which the data should be stored locally on the fog device instead of sending it to a software platform.	A local database should be added to the architecture in which the data can be stored. Given the current architecture it would be good to add a node which is responsible for writing and retrieving data to and from the database in the control node group.
3	A use case where other communication protocols than BLE (e.g., Thread, Matter, Zigbee, Z-Wave) are used for wearable communication.	In the sample node group a node for the other communication protocol should be added. [1]
4	A use case where long distance communication protocols (e.g., 4G, LTE-M, NB-IoT) are used for wearable communication.	In the sample node group a node for the long-distance communication protocol should be added. [1]
5	A use case in which the wearable buffers measurement data for a certain amount of time and only sends the collection of measurement data once in a certain interval.	Currently, the software architecture is designed to handle measurement data one at the time. Sending a collection of measurement data could cause problems. To solve this, the node in the sample node group which retrieves the collection of measurement data could temporarily store the data in the memory of the node and send it one by one to the control node.

Table 3 Scenario-based analysis of software architecture

[1] Currently, the chosen hardware for the proof of concept is a Raspberry Pi which supports Wi-Fi and Bluetooth (Low Energy). Using another communication protocol might require other hardware depending on the communication protocol.

Looking at the results of the scenario-based analysis the proposed software architecture is flexible enough to work with every scenario only requiring minimal changes. The only scenario which can cause problems is the fifth scenario in Table 3. The software architecture was designed for real-time processing of measurements one by one. By temporarily storing the data in the corresponding node in the sample node group and sending the measurement data one by one, the scenario would work. Nevertheless, this “solution” is not recommended because it is more a way of forcing the software architecture to work with that scenario by minimal changes rather than actually creating a good solution which might require bigger changes. It is more recommended to make a change in the implementation of the message structure, which is used to send messages between the nodes, so it supports messages containing multiple measurement data at once. In the transform node group, a sequencer transform node could be added which splits the multiple measurement data messages into single data messages if needed.

5. Discussion

This section discussed the most important findings. At the end of this section possible next steps for the future will be given.

5.1. Design iterations

The design phase of the project had several iterations of which the most important are described below.

5.1.1. Introduction of a Control node group

The biggest change in the design was the introduction of a control node group. In the first version of the design routing of the messages was done by the nodes of the sample, transform, process and software platform node groups which violates the principle of single responsibility. Single responsibility makes it easier to maintain and adjust the software [18] thus making it more flexible which is one of the research goals. It was decided to create a control node group with a routing node to handle the routing.

A second reason to add a control node group was for maintainability reasons. When there is a single node group which is responsible for controlling the gateway system, then logging and configuring the gateway system by the system administrator can all be done via that node group instead of having to access four separate node groups.

5.1.2. External and internal load balancer communication

The second change in the design is the communication between the internal and external load balancing nodes. The external load balancing node needs to make decisions based on information of the internal load balancing node. This means that they need to communicate with each other. The two options considered for this are shown in Figure 7.

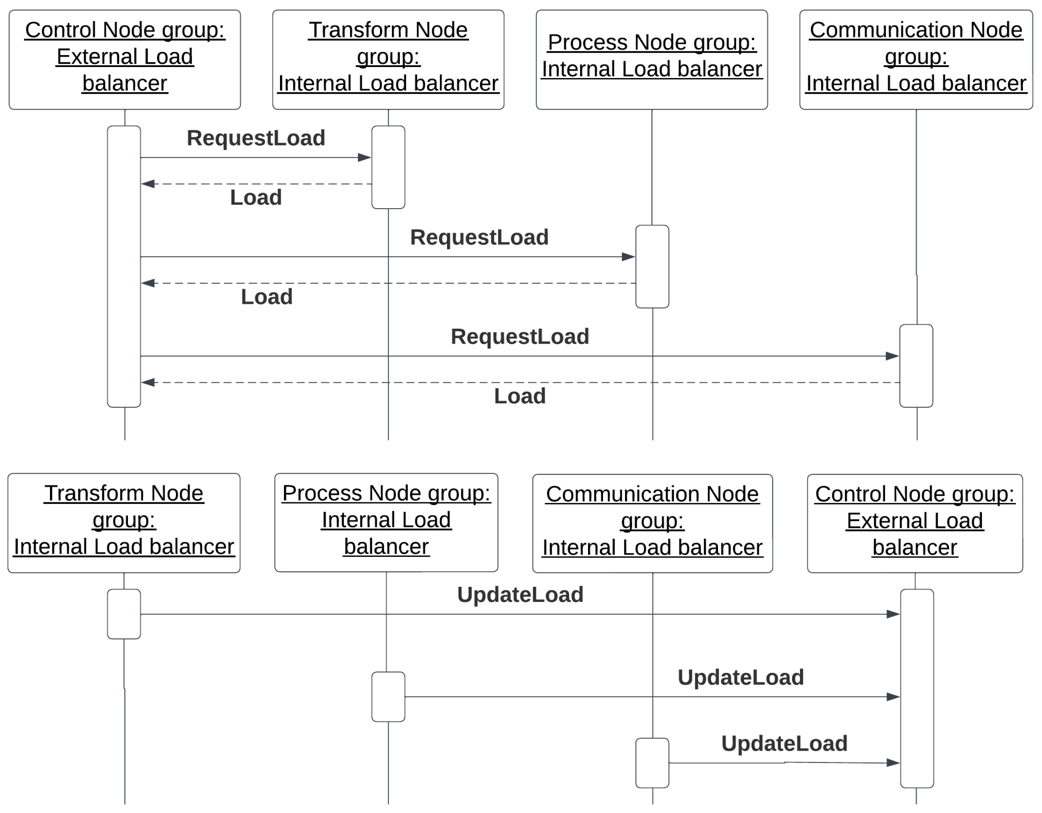
Control Node group: External Load balancer
Transform Node group: Internal Load balancer
Process Node group: Internal Load balancer
Communication Node group: Internal Load balancer
RequestLoad
Load
RequestLoad
Load
RequestLoad
Load
Transform Node group: Internal Load balancer
Process Node group: Internal Load balancer
Communication Node group: Internal Load balancer
Control Node group: External Load balancer
UpdateLoad
UpdateLoad
UpdateLoad

Figure 7 Two options External Internal load balancing node communication

The first option uses the polling approach meaning that the external load balancing node takes initiative. When the external load balancer needs to make a decision, it will send a request to all the internal load balancers to report their current load. When all the internal load balancers have replied to the external load balancer, the external load balancer can make a decision. The downside of this method is possible unnecessary communication. The external load balancer can request the load twice in a short period of time. When the load did not change in between those two requests the external load balancer is requesting a load which did not change compared to the previous request, thus creating unnecessary communication.

The second option uses the call-back approach meaning that the internal load balancer takes the initiative. On each change of the load the internal load balancer will send the change to the external load balancer. The external load balancer will have a call-back function to receive and temporarily store the load per node. When the external load balancer needs to take a decision, it checks the last stored value for each node instead of polling. Nevertheless, this approach also causes unnecessary communication since the internal load balancer is sending updates when the external load balancer does not need them yet. For example, as shown on Figure 8, the internal load balancers send the following loads: 2,3,4,3 and after the last 3 the external load balancer needs to check the load. The 2,3,4 which were sent were unnecessary since the external load balancer only needs to know the last 3. Because there was not a clear best option the first option is chosen for now with the reason that the external load balancer is the one who needs to know the loads so it should be in control.

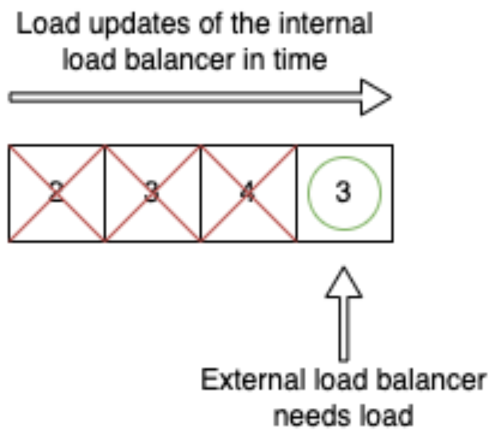
Load updates of the internal load balancer in time
2
3
4
3
External load balancer needs load

Figure 8 Unnecessary communication between internal and extern load balancing

5.1.3. Internal load balancing node

A third change to the design was the internal load balancing node. The original idea was to let the routing node send the message to the internal load balancing node of the node group and the internal load balancing node of the node group would redirect the message to the right node. This is also the way how it was implemented in the proof of concept. After giving it another thought, redirecting the message meant that it needed to be sent twice (From routing to load balancer and from load balancer to node). When sending the message twice the chances of mistakes are higher than sending it once.

Therefore, there was chosen to let the load balancing node return the ID of the node it needs to be sent to and let the routing send the message to the node with that ID directly. The number of messages is still two (Measurement message + Redirect measurement message or ID message + Measurement message) but the content of the ID message is way smaller thus using less performance and the odds of losing a measurement due to a faulty message is still decreased because the message is directly sent instead of twice via a redirect.

Due to the way the chosen communication protocol was implemented in the proof of concept, applying the new approach was too much work within the time left. Section 5.2. further elaborates on what the communication protocol implementation problem exactly is.

5.2. Proof of concept techniques

The techniques chosen in the proof of concept were purely chosen on how quick and easy they were to implement. This means that some of the techniques are not ideal and could be improved.

One of the chosen techniques is the TCP socket communication between the nodes. The experiences gained from the proof of concept, showed that socket communication, with the approach it is currently implemented with, in the proof of concept, works well for sending the messages between the routing and other nodes. Nevertheless, the implementation caused problems when load balancing was added to the proof of concept. TCP sockets uses device specific IDs for the nodes meaning that when an ID is requested and returned by the load balancing node this ID does not match the same device on the routing node. Figure 9 shows an example of the problem. If the routing (Device A) ask for the ID and the load balancer (Device B) returns the ID of the Node (Device C) because that is the node with the lowest load, the load balancer will send ID 3 which is the ID of the Node. The routing node is going to send the message to ID 3. The problem is that whilst on the load balancer ID 3 is the Node, the ID of the node on the routing is ID 4. ID 3 on the routing is the load balancer instead of the Node meaning that the routing is sending the message to the wrong ID.

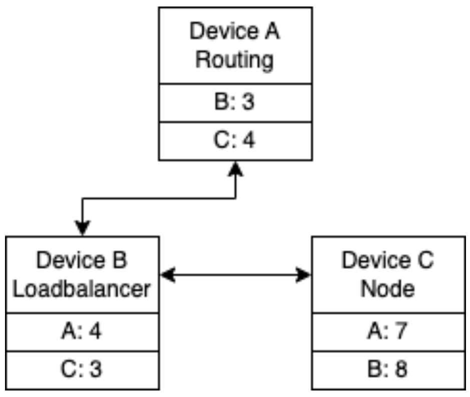

Figure 9 Device specific ID's example

Another technique which can be improved is the chosen programming language for the nodes. The proof of concept consists of a combination of C++ and Python nodes. For the proof of concepts, the sampling nodes for the MoveSense and the StressPatch were written in Python since the BLE library was easier to use. In terms of performance Python nodes are not ideal since python is a less performance efficient language in general compared to C or C++ [20] [21]. Writing nodes in Python will decrease the overall load a single fog node can handle. Writing all the nodes in C++ or C would therefore be more beneficial.

5.3. Flexibility

The overall experience of the design is that the modularity of having separate nodes for each individual responsibility worked well. While developing the proof of concept the number of nodes and functionality of nodes changed several times. This was easy to do, and no big practical problems were encountered. It was even possible to write nodes in different programming languages which was the case with the sampling BLE nodes which were written in Python instead of C++.

There were also no problems encountered when connecting different types of wearables to the gateway system. The approach of a separate sampling node made it easy to connect both the MoveSense and the Stresspatch and customize wearable specific things like which type of measurement data is extracted.

5.4. Routing node single point of failure

One possible problem of the design could be the routing node. When the routing node goes down the entire fog device does not work anymore meaning that the routing node is a single point of failure. A possible solution for this would be to always have a back-up routing node. When the routing node cannot keep up with the load or is unavailable the back-up node can help out. This is very similar to the load balancing concept designed for the other nodes. The only difference is that the two or possibly even more routing nodes have some shared information like the routes per sensor ID.

5.5. ROS2/Microservices

An interesting finding is the similarities between ROS2/Microservices and the proposed design. The overarching similarity of the proposed design, ROS2 and Microservices is that they all use a group of separate entities in which each entity has single responsibility. The similarity of specifically ROS2 and Microservices are discussed because the researcher was familiar with those two frameworks There is a possibility that there are more existing frameworks which also match the overarching similarity.

Looking at ROS2 first, ROS2 consists of separate nodes which have single responsibility. This is very similar to the approach of the proposed design. The single responsibility concept can also be found in Microservices. Each microservice is responsible for one specific task in a system. When this task is very demanding the capacity can be easily increased by launching another instance of the microservice. This is also the case with the nodes within the proposed design in which the load balancing node of a node group will launch a new instance. Seeing similarities in already existing and commonly used frameworks indicates that proposed design is a logical solution.

5.6. Future steps

There are several interesting next steps which are listed in the subsections below. These suggestions are not meant as a complete list.

5.6.1. Node communication research

As described in subsection 5.2., in the current version of the proof of concept socket communication was used to let the nodes communicate with each other. Nevertheless, this caused problems when trying to implement the load balancing functionality. The only reason why socket communication is chosen is because it was easy and fast to implement. When implementing the gateway system as an actual product instead of a proof of concept it is recommended to do research about which communication protocol is best suited for communication between the nodes.

5.6.2. Internal to external load balancing node communication

As explained in subsection 5.1.2. there are two possibilities of communication between the external and internal load balancing nodes. Because neither of the two options was clearly better than the other a possible next step would be to research which option works best. A possible approach would be to implement both options in a proof of concept to see which option works best in practice. Another and more recommended approach to do before implementing both options would be to make a list of pros and cons and use the system requirements to determine if certain pros are bigger advantages than other pros and if certain cons are smaller disadvantages than other cons.

5.6.3. Single point of failure possible solution

As pointed out in subsection 5.4. the routing node is a single point of failure. An important and highly recommended next step would be to explore the possibilities of solving this problem. A possible solution would be to have a back-up control node. Nevertheless, in the current design this was not taken into account and therefore, the design might need changes. A good first step would be to do an available product analysis to see if there are already existing solutions to avoid/resolve single points of failures. A quick google search resulted in finding a blog post [22] which lists methods for avoiding single points of failures which might be an interesting starting point.

5.6.4. Load balancing and scalability

Load balancing and scalability is a complicated topic and due to limited time, it was only briefly covered in the project. The software architecture contains load balancing. Nevertheless, this is more on architecture level and not really on implementation level. Functionalities like deciding when to send a message to a certain node and when to launch or shutdown nodes, are not research yet. A quick search on google scholar with the query “load balancing” shows a lot of publications [23], [24], [25], [26], [27], [28], [29] about load balancing approaches and algorithms. This indicates that there are already existing solutions. A recommend first step would therefore be to do an available work analysis about load balancing and scaling. An interesting publication was found [30] about scaling of latency-sensitive applications which could be an good starting point.

5.6.5. Wearables with other communication protocols

The wearables used within the proof of concept all use BLE as communication protocol. Due to limited time and an unfortunate cooperation with one of the partners within the SWSP project wearables which use another communication protocol than BLE were not implemented. This would be an important future step to further verify the possibility and easiness of using different types of wearables with the gateway device.

5.6.6. New approach internal load balancing

As described in subsection 5.1.3. the new approach for internal load balancing was not implemented in the proof of concept. A future step would be to implement this new approach in a proof of concept to verify that there are no mistakes or missing parts in the approach.

6. Threats to validity

We will consider several threats to validity of the conclusions in the discussion.

The first threat to validity is the timeboxing of the available work analysis. Due to limited time the available work analysis was only limited to the first page of google scholar. It is possible that on next page(s) literature can be found which can give better or other information than the literature of the first page. Nevertheless, reading the abstracts of the first four hits of the second page [31] [32] [33] [34] gave the indication that those publications were not giving new or relevant information compared to the already analysed publications.

The second threat to validity is that only the parts of the design that are vital for a minimal viable product were implemented in a proof of concept meaning that the design is not fully verified. This leaves possible mistakes or flaws in the proposed solution untested. Nevertheless, the essential parts of the gateway system were implemented with an exception for external load balancing and scaling which is discussed in the next paragraph. Because only non-essential parts of the design were not implemented it is likely that even if there are mistakes in those parts the needed changes are minor. The design was also reviewed by two other researchers and an external expert who did not find flaws in the design.

The third threat to validity is that due to limited time the external load balancing approach was not implemented in the proof of concept meaning that it is not verified. This is a threat to validity because this is an essential part of the design. If, with the proposed design, it is not possible to load balance externally the system might not be able to handle the required load during high data interval situations which was one of the aspects described in the use case definition. However, the internal load balancing is implemented, and because external load balancing is similar to internal load balancing no major problems are expected. The design was also reviewed by two other researchers and an external expert who did not notice any problems in the design.

The fourth threat to validity is that the new approach of the internal load balancing is not implemented in the proof of concept. Therefore, it is possible that the new approach is not flawless. The fact that the design was reviewed by two other researchers and an external expert who did not foresee any problems gives enough confidence that possible found flaws will be minor.

The fifth threat to validity is the limited implementation of web interface nodes for both real time data monitoring, logging and configuration. It could be possible that the design of this part is not detailed enough thus causing missing information in the design. Nevertheless, the design has been reviewed by two other researchers and an external expert which gives enough confidence that there are no major problems in this part of the design.

The sixth threat to validity is the scaling of nodes. Due to limited time and problems with the chosen communication protocol scaling was not implemented in the proof of concept. Therefore, it is possible that the design of this part is not flawless. Nevertheless, the concept scaling is used in many software applications including microservices in which Kubernetes [35] can be used to scale nodes up or down. This gives enough confidence that there is enough available work about this topic to implement it. The design also has been reviewed by two other researchers and an external expert.

The seventh threat to validity is that due to limited time only BLE wearables are connected in the proof of concept. In theory it should not matter which communication protocol the wearable uses to send its data because the responsibility of the sampling node is to extract the data and put it into the defined message format of the software architecture design. Nevertheless, the design has been reviewed by two other researchers and an external expert which gives enough confidence that there are no major problems in this part of the design.

The eighth threat to validity is that during the scenario-based analysis of the software architecture, due to limited time, only base functionalities and possible vulnerable points of the design were kept in mind while drafting the scenarios. Because of this it is possible that there are potential missing scenario's which can undermine the validity of the flexibility of the proposed design. Because base functionalities and possible vulnerable points were the focus of the scenario it is unlikely that problems which will be identified by other scenarios will case very big problems.

7. Conclusion

In this report a technical design is proposed of a gateway system architecture for wearables. A fog architecture approach was used in which the measurement data of multiple different types of wearables can be collected, the measurement data can be, if needed transformed and processed, and sent to a software platform. To make the design flexible a design software architecture pattern was used.

The research resulted in the following most important outcomes:

- A software architecture for a gateway system consisting of separate nodes with all their own single responsibility. The architecture consists of four layers; retrieving the measurement data from the wearables, transforming the data, applying algorithms or AI on the data, sending the data to a software platform in the cloud or a real time monitoring dashboard. The software architecture is able to scale up or down depending on the demands.
- A proof of concept showed that the designed architecture is implementable and works for the main use case of the research.
- A scenario-based analysis verified that the software architecture was flexible.
- An observation which showed similarities between the proposed design and the already existing frameworks ROS2 and Microservices were noticed. Indicating that the approach of the design is logical.

- A potential bottleneck caused by the routing node, which is responsible for the routing of the messages between the different nodes, being identified as a single point of failure.

The research created some interesting next steps for the future. One of the most important next steps would be to research the chance and impact and, if needed, find a solution for the routing node being a potential bottleneck within the design. When the routing node turns out to be a bottleneck the impact of this could be really high because the system is not able to execute any functionality without the routing node. Another important future step would be to add a wearable which uses another communication protocol than BLE to the proof of concept. This is an extra verification that the proposed design can not only handle multiple different types of wearables which measure something different, but also can handle wearables which use a different communication protocol.

Appendixes

Appendix	Document Name
Appendix A	Research document Section 2.1. Use case definition
Appendix B	Research document Section 3.2. Available work analysis
Appendix C	Design document

Bibliography

[1] Movesense, “Movesense Sensor HR2,” Movesense, n.d.. [Online]. Available: https://www.movesense.com/product/movesense-sensor-hr2/. [Accessed 12 January 2024].

[2] Empatica, “EmbracePlus,” Empatica, n.d.. [Online]. Available: https://www.empatica.com/en-gb/embraceplus/. [Accessed 12 January 2024].

[3] M. K. Sebas Bakker, “Stress Wearables - Project plan,” Eindhoven, 2023.

[4] HBO-I, “Realise as an expert,” n.d.. [Online]. Available: https://ictresearchmethods.nl/patterns/realise-as-an-expert/. [Accessed 19 December 2023].

[5] HBO-I, “Available product analysis,” n.d.. [Online]. Available: https://ictresearchmethods.nl/library/available-product-analysis/. [Accessed 19 December 2023].

[6] HBO-I, “IT architecture sketching,” n.d.. [Online]. Available: https://ictresearchmethods.nl/workshop/it-architecture-sketching/. [Accessed 19 December 2023].

[7] HBO-I, “Brainstorm,” n.d.. [Online]. Available: https://ictresearchmethods.nl/workshop/brainstorm/. [Accessed 19 December 2023].

[8] HBO-I, “Prototyping,” n.d.. [Online]. Available: https://ictresearchmethods.nl/workshop/prototyping/. [Accessed 19 December 2023].

[9] R. Kazman, “Scenario-Based Analysis of Software Architecture,” *IEEE Software,* vol. 13, no. 6, pp. 47-55, November 1996.

[10] J. Kaur, “What's the Difference Between Cloud, Edge, and Fog Computing?,” June 2023. [Online]. Available: https://www.xenonstack.com/blog/cloud-edge-fog-computing. [Accessed 6 November 2023].

[11] DIGITEUM TEAM, “Difference between Cloud, Fog and Edge Computing in IoT,” 4 May 2022. [Online]. Available: https://www.digiteum.com/cloud-fog-edge-computing-iot/. [Accessed 6 November 2023].

[12] A. Overheid, “Understanding Fog Computing vs Edge Computing,” 15 April 2022. [Online]. Available: https://www.onlogic.com/company/io-hub/fog-computing-vs-edge-computing/. [Accessed 6 November 2023].

[13] WinSystems, “Cloud, Fog and Edge Computing – What’s the Difference?,” 4 December 2017. [Online]. Available: https://www.winsystems.com/cloud-fog-and-edge-computing-whats-the-difference/. [Accessed 6 November 2023].

[14] M. S. Diab, “Embedded Machine Learning Using Microcontrollers in Wearable and Ambulatory Systems for Health and Care Applications: A Review,” *IEEE Access,* vol. 10, pp. 98450-98474, 15 September 2022.

[15] A. Ilyas, “Software architecture for pervasive critical health monitoring system using fog computing,” *Journal of Cloud Computing,* vol. 11, p. 84, 30 November 2022.

[16] A.-T. Shumba, “Leveraging IoT-Aware Technologies and AI Techniques for Real-Time Critical Healthcare Applications,” *Sensors (Basel),* vol. 19, p. 7675, 10 October 2022.

[17] J. E. Bardam, “Software Architecture Patterns for Extending Sensing Capabilities and Data Formatting in Mobile Sensing,” *Sensors,* vol. 22, p. 2813, 6 April 2022.

[18] R. C. Martin, “The Single Responsibility Principle,” 8 May 2014. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html. [Accessed 12 January 2024].

[19] S. Bakker, “An Architectural Framework for a Stress Wearable Data Platform,” Eindhoven, 2024.

[20] L. Prechelt, “An Empirical Comparison of Seven Programming Languages,” *Computer,* vol. 33, no. 10, pp. 23-29, October 2000.

[21] S. Ali, “A Pragmatic Comparison of Four Different Programming Languages,” *ScienceOpen,* 21 June 2021.

[22] M. Otta, “Avoiding Single Points of Failures in Distributed Systems,” 26 October 2023. [Online]. Available: https://www.baeldung.com/cs/distributed-systems-prevent-single-point-failure. [Accessed 12 January 2024].

[23] E. J. Ghomia, “Load-balancing algorithms in cloud computing: A survey,” *Journal of Network and Computer Applications,* vol. 88, no. 1084-8045, pp. 50-71, 8 April 2017.

[24] T. Bourke, Server load balancing, n.d..

[25] S. Sharma, “Performance Analysis of Load Balancing Algorithms,” *World Academy of Science, Engineering and Technology International Journal of Civil and Environmental Engineering,* vol. 2, no. 2, 2008.

[26] Y. Azar, “On-line load balancing,” in *Online Algorithms. Lecture Notes in Computer Science*, Heidelberg, Springer, Berlin, Heidelberg, 2005.

[27] A. Khiyaita, “Load balancing cloud computing: State of art,” *2012 National Days of Network Security and Systems,* pp. 106-109, 26 July 2012.

[28] J. Watts, “A practical approach to dynamic load balancing,” *IEEE Transactions on Parallel and Distributed Systems,* vol. 9, no. 3, pp. 235-248, March 1998.

[29] T. Chou, “Load Balancing in Distributed Systems,” *EEE Transactions on Software Engineering,* Vols. SE-8, no. 4, pp. 401-412, July 1982.

[30] F. Rossi, “Hierarchical Scaling of Microservices in Kubernetes,” in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, Washington, DC, USA, 2020.

[31] S. Boovaraghavan, “Mites: Design and Deployment of a General-Purpose Sensing Infrastructure for Buildings,” *Association for Computing Machinery,* vol. 7, no. 1, 2023.

[32] S. Palomeque-Mangut, “Wearable system for outdoor air quality monitoring in a WSN with cloud computing: Design, validation and deployment,” *Chemosphere,* vol. 307, no. 3, 2022.

[33] B. Ahamed, “Design of an energy-efficient IOT device-assisted wearable sensor platform for healthcare data management,” *Measurement: Sensors,* vol. 30, 2023.

[34] I. Shabani, “Design of a Cattle-Health-Monitoring System Using Microservices and IoT Devices,” *Computers,* vol. 11, no. 5, p. 79, 2022.

[35] L. A. Vayghan, “Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, USA, 2018.