TEAM PROJECT FINAL REPORT

Software Architecture – Sea Buoy

TUTORIAL 2 GROUP 9 - TEAM MEMBER

Tamsin Low, 13925969

Nexus Baquir, 13572324

My Duong, 13934070

Wei Ming Edward Ong, 14005817

Sakura Adachi, 24648418

Andrew Soen, 14189043

Table of Contents

Report Summary

This report will discuss the software architecture of a case study for the University of Technology Sydney as per the assessment task. The case study is about the utilization of free-floating sea buoys to provide navigation and weather data for air and ship traffic at sea. The sea buoys should also be able to initiate an SOS broadcast in times of distress and be equipped with a red light to aid in sea-search operations.

The following report will cover the project context which includes the system's requirements, user stories and narratives, risks and risk mitigation, and constraints. Additionally, there are initial and alternatives designs for the conceptual, execution and implementation architecture; each architecture will be thoroughly evaluated before the final architecture is chosen.

Project Context

System Purpose and Objectives

The Maritime Safety Bureau owns a collection of free-floating sea buoys that provide navigation and weather data to air and ship traffic at sea. The purpose of the system is to facilitate this navigation and weather data collection and communication on each sea buoy. Additionally, the system will execute any appropriate instructions that are used for maintaining the safety of the seas.

There are several important objectives that the system is to achieve in addition to its' primary function of collecting and providing data to passing vessels at sea, including: broadcast the obtained data to passing vessels upon request, initiating sea-buoy for data collection every 10-30 seconds, broadcasting general information every 60 seconds, allowing modification so that users can add new functions and change its' priority, and operating for an extended period of time with minimal maintenance. Alternatively, sailor enables to activate and deactivate the sea-buoy's red light to signify when there is a passing vessel; when the emergency switch is turned on, the system will trigger the nearby sea-buoy to continuously broadcast the SOS signal until the switch is reset.

Assumption

The teams have made a few assumptions based on the selected case study, such as all passing vessels have installed radio communication devices and the teams will only need to connect them to the implemented system. It is also assumed that the sea-buoys are equipped with a functional radio antenna that enables to transmit and receive signals, even though it may not have a red-light. Also, the sea-buoy are believed to be configurable, enabling the development teams to effectively reconfigure and interconnect the sea-buoys with the designed system.

System Functional and Non-Functional Requirements

Below is a list of simplified functional and non-functional requirements; there are in-depth explanations of these requirements in each of the architecture designs:

Functional Requirements	*Non-Functional Requirements*
Communication Management – refers to the interaction between the radio, sea-buoys and passing vessels	*Maintainability and Availability – the system is to operates for an extended period without maintenance*
Event Management – respond to the broadcast, red-light switch, and SOS emergency signal broadcast	*Security – prevent unauthorised user access or modification that may cause major issues to passing vessels operations*
Data Collection – collect and process different data	*Flexibility and reliability – the ability of system to recover or maintains its operation when issues arise (e.g., corruption or disruption in transmission)*
Database Management – store and retrieve data upon request	*Performance – the actions must be completed within the restricted time frame to ensure no delay providing data to stakeholders*

Stakeholders

The table below outlined the primary and secondary stakeholders along with a description of their interest and roles as well as the identifying the appropriate system qualities for each stakeholder.

Stakeholder	Primary/Secondary	Interest	Qualities
Sailor	Primary	• Needs to send a continuous SOS signal • Turns off SOS signal as help has arrived	• Performance • Reliability
Maritime Safety Bureau	Primary	• Responsible for maintaining safety on the seas - includes reporting environmental conditions and saving lives • Owners of the Sea buoys	• Scalability • Modifiability • Portability
Data Analyst	Primary	• Monitors and analyses weather data • Particularly dangerous weather patterns such as storms and hurricanes • Provides navigational information for ships (including search and rescue crews) • Provides reports on weather and climate patterns and warnings (E.g., storms, La Nina, climate change) • Provides reports on sea traffic activity (E.g., number of successful/unsuccessful rescues, areas with high volumes of ship traffic) • May require new types of data for further weather analysis which may lead to new sensors on the sea buoys (E.g., pressure sensors, water height measures), or the installation of new sea buoys	• Performance • Reliability • Data Integrity • Modifiability
Developer	Primary	• Build the software for the Sea Buoy's data collection and data communication system • Test and maintain the software	• Modifiability • Testability • Reliability
Maintenance Crew	Primary	• Install the required equipment and the developed software on the sea buoys • Performs maintenance and repairs on the Sea buoy, including the equipment	• Reliability • Usability
IT Support	Primary	• Responsible on maintaining the operation and stability of the Sea Buoy software system • Troubleshoot any software and technology problems	• Reliability • Testability

Ships	*Primary*	• *May need rescuing* • *Want to know current weather conditions and warnings* • *Want to know best navigation paths* • *Want to know the sea traffic conditions*	• *Reliability* • *Performance* • *Usability*
Maritime Emergency Personnel	*Primary*	• *Respond to SOS signals* • *Perform search and rescue operations* • *May need to navigate hazardous weather conditions*	• *Reliability* • *Performance*
Government	*Secondary*	• *Want to know the safety and traffic trends of the sea* • *Allocate funding to the Maritime Safety Bureau*	• *Reliability*
General Public	*Secondary*	• *May engage in various sea activities. E.g., fishing, sailing, deep sea diving* • *Need to know the weather and warnings so they can safely participate in these activities*	• *Reliability*
Equipment Suppliers	*Secondary*	• *Provide physical sea buoys, sensors, radios, and other necessary equipment* • *May assist with installation and ensure equipment compatibility*	• *Modifiability* • *Usability*

User Stories

User Story #	As a/the	I want to...	So that...	Priority (low/medium/high)
101	Sailor	Send a SOS broadcast	I can get help in emergency	High
102	Sailor	Stop the SOS broadcast	I can terminate the help request once I have received help	High
103	Sailor	Get a 24-hour summary	I can view the information from the last 24 hours	Medium
104	Sailor	Turn on the red light	I could signify that there is a ship approaching	Medium
105	Sailor	Turn off the red light	I could signify that the ship is moving away from the sea buoy	High
201	Data Analyst	Get the average wind and temperature readings	I can determine the weather conditions and patterns.	Medium
202	Data Analyst	Get location readings	So that I know where the data came from.	Low
301	Maritime Safety Bureau	Add sensor(s) to a sea buoy	New data can be collected	Medium
302	Maritime Safety Bureau	Remove sensor(s) from a sea buoy	Only the most relevant data is collected.	Low
303	Maritime Safety Bureau	Change the broadcast priorities	The most relevant information is received first	Low
401	Maintenance Crew Member	Be able to be notified for urgent repairs	I can act immediately and solve critical issues	High
402	Maintenance Crew Member	Be able to receive the collected data	I can assess when to send out the crew member	Medium
501	IT Support	Be notified of lost connectivity with devices (sensors, red light, emergency switch, radio)	The peripheral can be fixed.	Low
601	Maritime Emergency Personnel	Receive a SOS broadcast	I can save those who need help.	High
701	Ship Crew	Receive the periodic location information	We know our location	Medium
801	Developer	Configure the sea buoy's sensors	The system can receive readings at different times	High
802	Developer	Configure the sea buoy's radio transmitter	I can send automatic broadcasts	High
803	Developer	Configure the sea buoy's radio receiver	I can receive requests from sea vessels	High
804	Developer	Change the sea buoy's sensor configurations	The cadence of the readings is update to date with changes in the sensor's cadences	Low
805	Developer	Change the broadcast priorities	The broadcast transmission order reflects the changes in priorities	Medium
806	Developer	Change the periodic broadcast frequency	The periodic broadcast frequency is up to date.	Low

User Narrative

These user narratives provide a perspective from some of the key stakeholders of this project and help to expand on some of the user stories as well as the system qualities they deem significant.

Maritime Safety Bureau

We are responsible for maintaining safety on the seas. This includes monitoring and providing warnings and safety advice as well as responding to calls for help and conducting search and rescue operations to save lives. We have hundreds of Sea Buoys scattered around the waters to provide us with the information necessary to maintain the sea's safety. We cannot afford to have frequent maintenance performed on each of the sea buoys so we would like it if the system is reliable and stable. Not every sea buoy is the same as they might have different sensors and might not even have a red light. Additionally, the sensors on the Sea Buoy's may be adjusted according to our data gathering needs. The system should be flexible to these changes and be compatible with the variability amongst our Sea Buoys.

Sailor

If we are in an emergency, we will try to find a sea buoy so we can send an SOS message. Once we reach the sea buoy, I can flick the emergency switch on to initiate the SOS broadcast. We need the SOS message to be continuously sent until another vessel or some help comes. Once support has reached us, we generally reset the emergency switch so we can stop sending SOS broadcasts as it is no longer required.

Data Analyst

As a data analyst at the Maritime Safety Bureau, I monitor and analyse weather data and patterns and assist in providing information for search and rescue operations. Some of my responsibilities include identifying dangerous weather hazards such as hurricanes and storms, providing ship crews (including search and rescue crews) with emergency signal locations and safe navigation paths, and monitoring the impact of climate change on the oceanic environment. To carry out my responsibilities I use the weather and locational information received from the numerous sea buoys that are based in various locations. I need this information to be correct and updated regularly so that my findings and reports are accurate. Currently, every minute is a suitable frequency for getting the most up to date information. I only need to wind and temperature averages and the location data, however, in future, I may require more types of data to better understand the weather patterns, such as pressure readings and weather height readings, so it would be good if the system could provide this.

Emergency Personnel

When we receive an SOS signal, we need to know the exact location that the SOS signal came from, so we can go and rescue those who require our help. We also like to know the current environmental conditions at sea, particularly near the SOS signal so that we predict the emergency situation and plan our rescue operation. The analysts can give us the location and environmental information and even provide us with a safe navigation path. In other situations, such as search and rescue operations, no SOS signal may have even been involved. In this case analysts and members of our team use navigation data to guide us where to conduct the search and rescue operation.

Maintenance Crew

Conducting maintenance on the sea buoys takes a lot of effort, because I have to sail directly to the sea buoy. I don't usually conduct regular check-ups on the sea buoy, so I would like to only go to check the sea buoy if it is urgently required. Being notified of this would be helpful, so I can assess the urgency of the fault and go and conduct maintenance.

Risk and Risk Mitigation

Risk	Description	Mitigation
	Project Development	
Requirement misinterpretation	The interpretation of the business requirements is incorrect	Regular communication between the stakeholders to receive feedback and clarity on the design and requirements
Changing scope	The scope may be incomplete, or drastically expand, in which case the project will have to be re-evaluated and delayed.	Allocate more time for changing scope. Ensure iterative development is implemented so changes in scope are identified as early as possible.
Slow Development	The development of the project extends past the originally planned timeline, which may incur more cost.	Have a delay buffer when planning timelines. Ensure communication of timelines and development progress is frequent, so any delays can be identified early and timelines can be adjusted accordingly.
Lack of resources	The development team does not have enough resources to progress efficiently	Ensure there are adequate resources when planning and contact stakeholders if more are required.
Team friction	The development team and management do not operate well together, due to differing ideas or personality clashes. This may further delay progress and impact the quality of the software.	Ensure team wellbeing is prioritised with regular check-ins with the team. Encourage team interaction and team building activities to boost morale and communication.
	System Implementation Risks	
Software incompatibility	Updates to software modules for security and new feature purposes, may cause compatibility issues.	Buy components that have generic purposes and build components that have more specific purposes to the system
Hardware incompatibility	The hardware modules become obsolete over time	Use hardware modules that are system standards
Lack of software quality	May introduce more bugs and make it difficult to maintain and modify	Ensure best coding practices are used when developing the software. Use rigorous testing is used to ensure quality standards.
	System Performance	
Environmental Damage	The physical hardware on the system is damaged due to environment factors such as water leakage.	Ensure the system located in a secure position on the sea buoy and is not exposed to the open environment.
Broken peripherals	The peripherals are faulty so the system cannot collect or send any data.	Conduct maintenance on the sea buoy and try integrating fault notifications in the system.
Incorrect location information	Incorrect or lack of location information from the sea buoy may hinder ships' ability to physically navigate to the buoy	Provide a list of sea buoy and its location to sailor so if the system failed to 'search' for the sea buoy, sailor still could refer to the location written in the list and travel to the destinated sea buoy
Volume Overload	The system crashes when faced with high volumes of data traffic.	Ensure the system is stress and volume tested. Establish a minimum threshold for the system's performance requirements.
Inadequate data storage	The system cannot store all the readings data, resulting in data loss.	Ensure old data is periodically deleted and test how much storage space is required. Purchase this required storage space and more for a buffer.

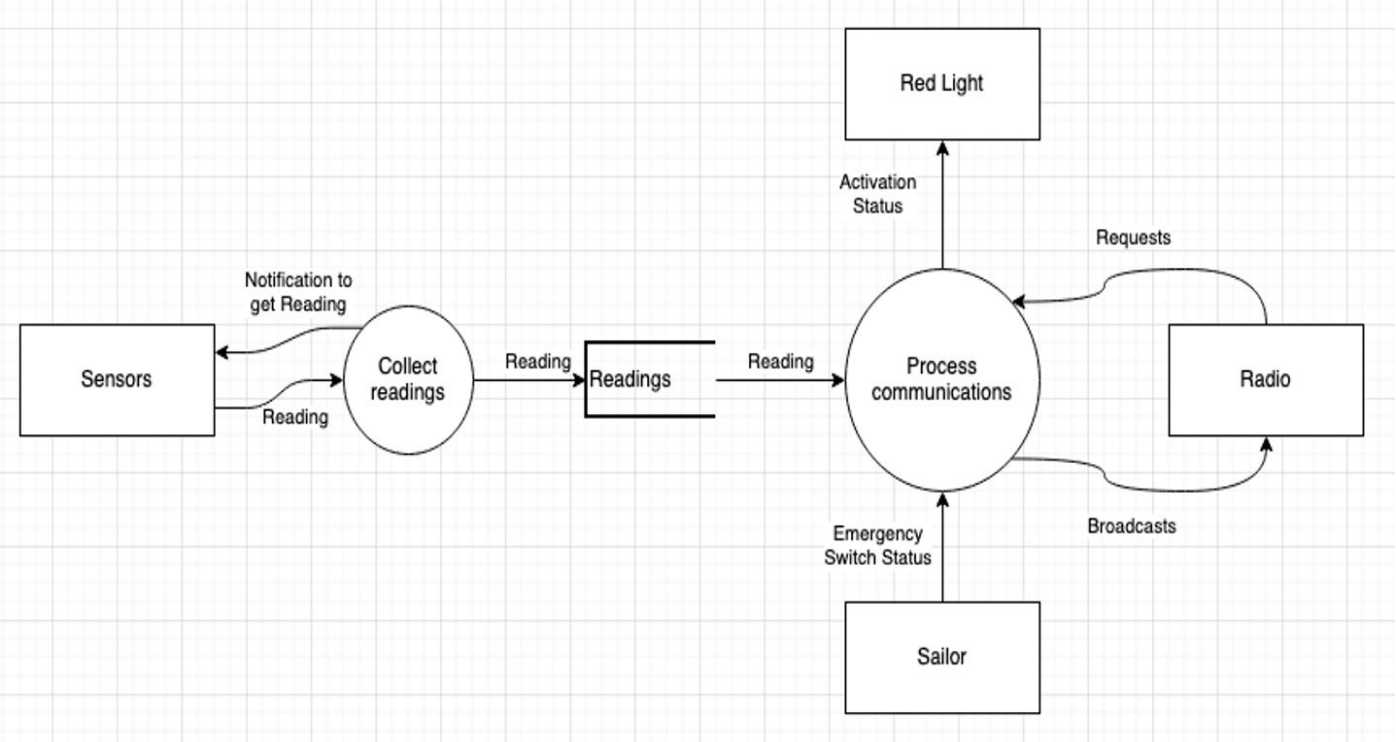
Constraint	Explanation
Limited source to power	*The remote location of the sea buoys means that access to power is reduced to either battery or solar sources, where the luxury of connecting to the grid is unavailable.*
No internet access	*The remote location of the sea buoys means that there is no internet access on the sea buoy. Using any cloud architecture will be difficult considering it needs access to the internet. Embedding the system onto the sea buoy is a more viable option.*
No user interface	*Users interact with the system using radio communication. The transmission and reception of these radiograms resulted in no user interface.*
Constrained memory	*Storing large files could be difficult considering no internet access and memory will have to be provided on the sea buoy itself.*
Budget	*The Maritime Safety Bureau have expressed that cost is a concern due to the large scale of sea buoys the system will be applied on. This limits the amount of project resources and the solution's implementation and maintenance costs.*
Delivery Time	*The project will be expected to be delivered according to the determined schedule. This reduces development time and increases the risk of delays.*

Conceptual Architecture

Initial Conceptual Architecture and Explanation

The system's conceptual architecture uses a context diagram because it easy for all stakeholders to interpret and understand the overall system. It demonstrates how the main entities and processes interact together.

The system's main processes are to collect readings and process communications. The sensors, sailor, red light, and radio are the main entities identified in the system that interact with the two main processes. The architecture contains one data store to temporarily store the readings used for the broadcasts.



The following explains the main scenarios of the context diagram that are used to assess against the system's functional requirements

- *Sensors provide readings based on their cadences*
- *Readings are stored in the data store*
- *Readings are then processed into a broadcast message and sent to the radio to transmit.*
- *Requests received from the radio are processed, this includes that may activate/deactivate the red light and historical reading requests*
- *The emergency switch is turned on by a sailor which triggers a SOS broadcast*

Finalised Conceptual Architecture and Explanation

From the peer review feedback, it was highlighted that the initial conceptual architecture did not provide enough detail of the system, which hindered overall understanding. Consequently, more entities were identified in the final conceptual architecture as well as their relations with each other.

Icons were used to illustrate the entities that define peripheral devices or actors. To represent the timing specifications (806), the timer entity is included. Here, it is related to the sensor controller since the sensor readings are received based on their cadences. Additionally, there is a relation between the timer and the periodic broadcast since these broadcasts also have a set cadence. The environment and ship entities were also included to demonstrate how the system interacts with its end users.

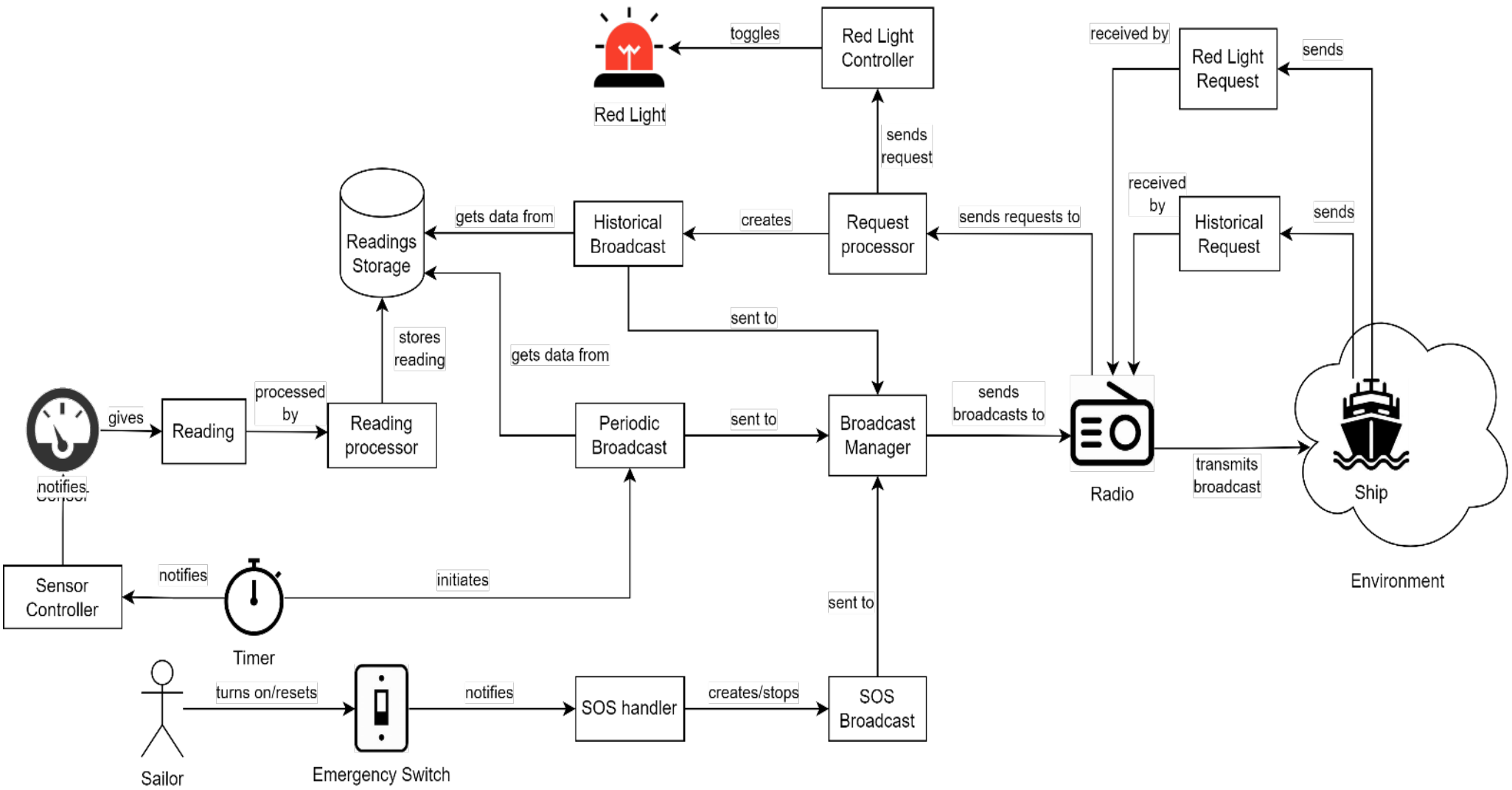
The Broadcast manager was also added where it is responsible for prioritising the three types of broadcasts (historical, periodic and SOS broadcast) for User Story 304. The SOS handler continually oversees the creation of the SOS broadcast until it gets the reset status from the emergency switch. With the request processor, this would process the raw requests received from the radio so its instruction can be interpreted.

Two controller entities were added as they are relevant when communicating with some of the peripheral devices, such as the red light and sensor.

Entities were also added for broadcasts. Some of these broadcasts were related to the readings storage to show they contained readings in their broadcast. The SOS broadcast can be viewed as the result of the sailor turning on the emergency switch, satisfying the Sailor's User Narrative. Similarly, entities were added for demands to differentiate the varying types of requests sent by the ship.

The different sensor types were not added as separate entities. This is because they are all covered under the sensor entity as they all fundamentally provide the same function and have the same relations with the same entities. This being their controller notifies them to provide a reading to the reading processor

This added detail and comparison against some of the user stories and narratives clarifies how the conceptual architecture satisfies the system's functional requirements.

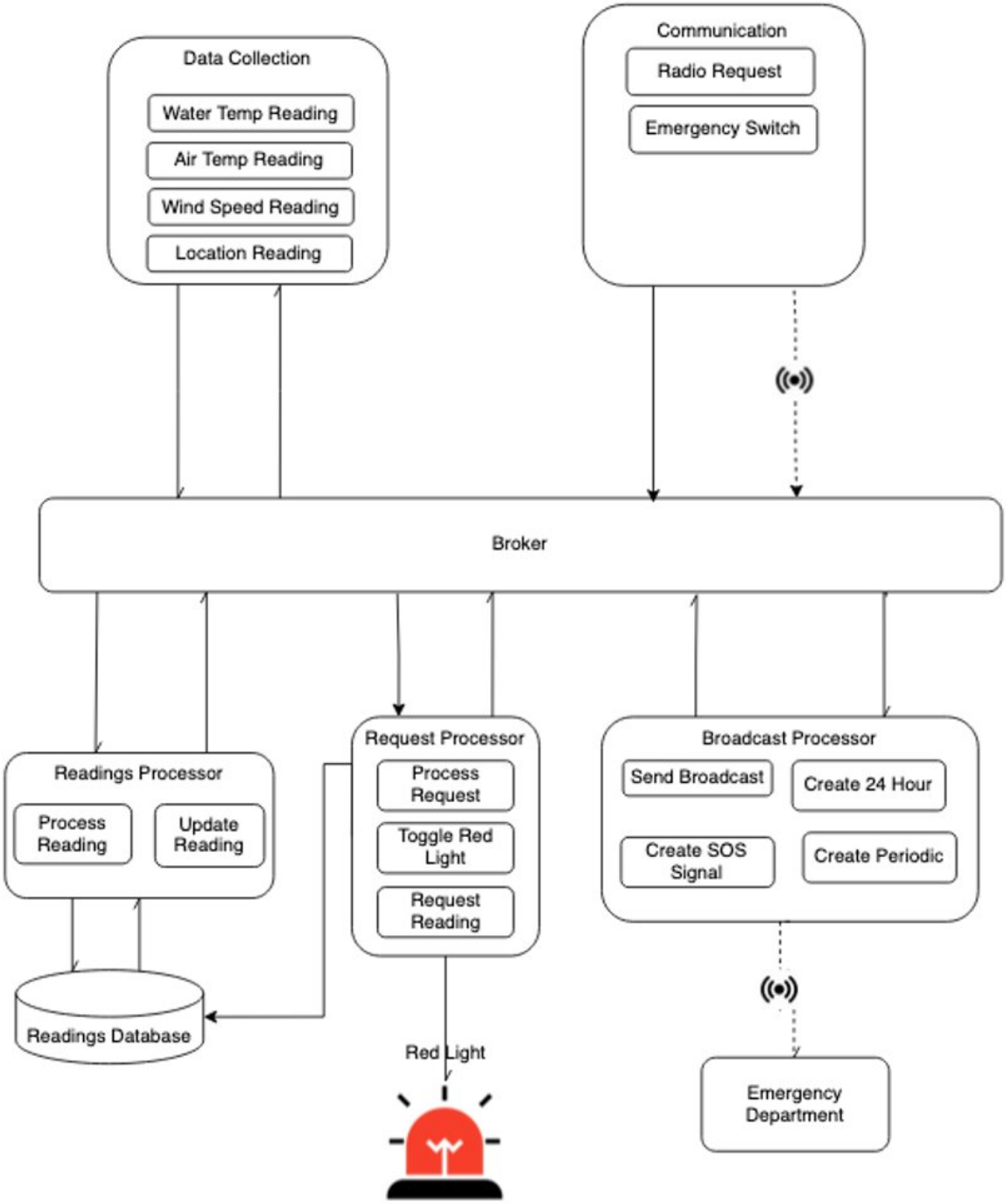

Red Light
toggles
Red Light Controller
received by
Red Light Request
sends
sends request
received by
Historical Request
sends
gets data from
Historical Broadcast
creates
Request processor
sends requests to
Readings Storage
sent to
stores reading
gets data from
gives
Reading
processed by
Reading processor
Periodic Broadcast
sent to
Broadcast Manager
sends broadcasts to
Radio
transmits broadcast
Ship
notifies
Environment
Sensor Controller
notifies
initiates
sent to
Timer
Sailor
turns on/resets
Emergency Switch
notifies
SOS handler
creates/stops
SOS Broadcast

Execution Architecture

Initial Execution Architecture and Explanation

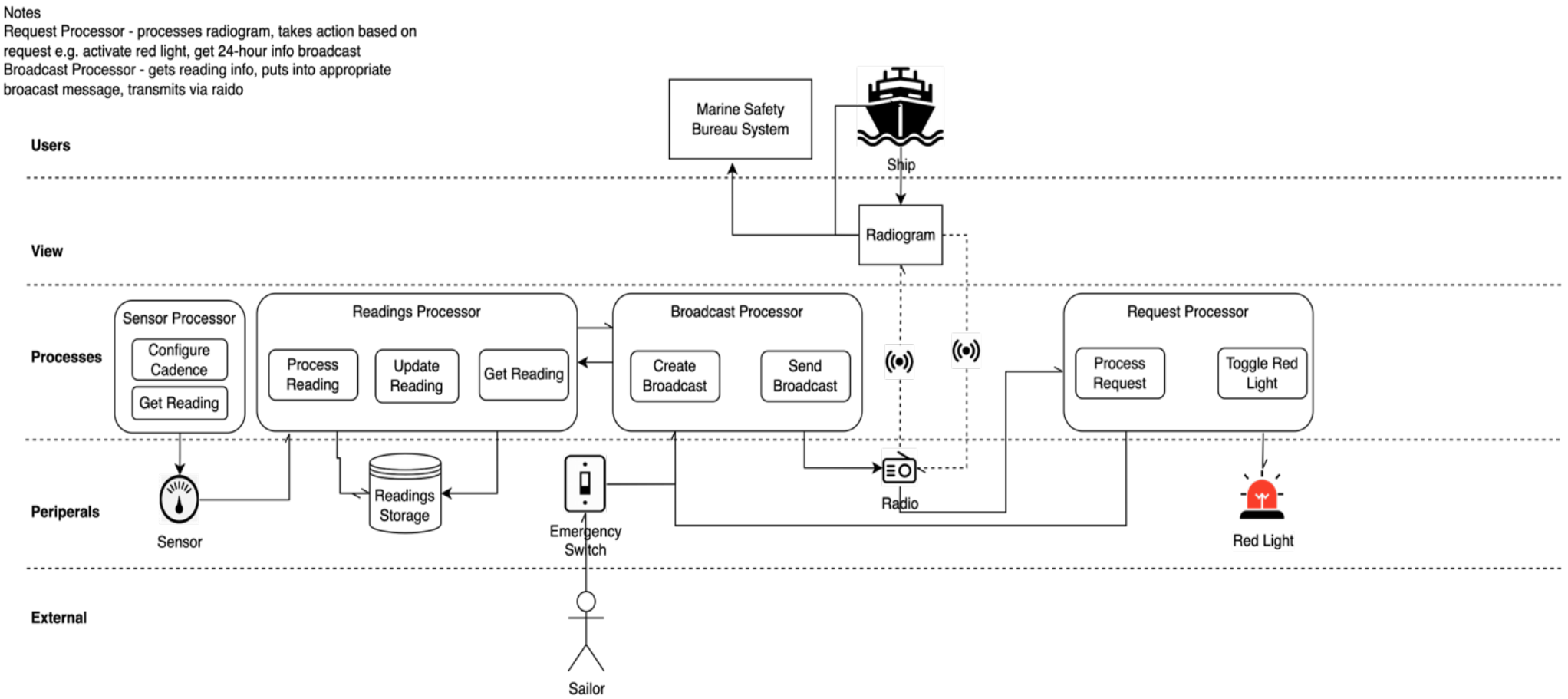
The initial execution architecture used the broker architecture pattern. At runtime, the broker component coordinates requests and responses between clients and servers. The broker has the details of the servers and the individual services they provide. The clients send requests, and the broker finds the right server to route the request to. Should there be a need, the broker will send the responses back to the clients.

Here, the clients consist of the readings, radio requests (red light and historical) and the emergency switch status. This architecture was initially chosen because of its loosely coupled design, as most of the system was connected to the broker, and the flow of data was simple to understand. However, this architecture does not show data control in terms of prioritisation and event orchestration. Additionally, the asynchronous communication with the broker is not ideal, considering this is a real-time embedded system where timing constraints matter.



1st Alternative Architecture Pattern – Layered

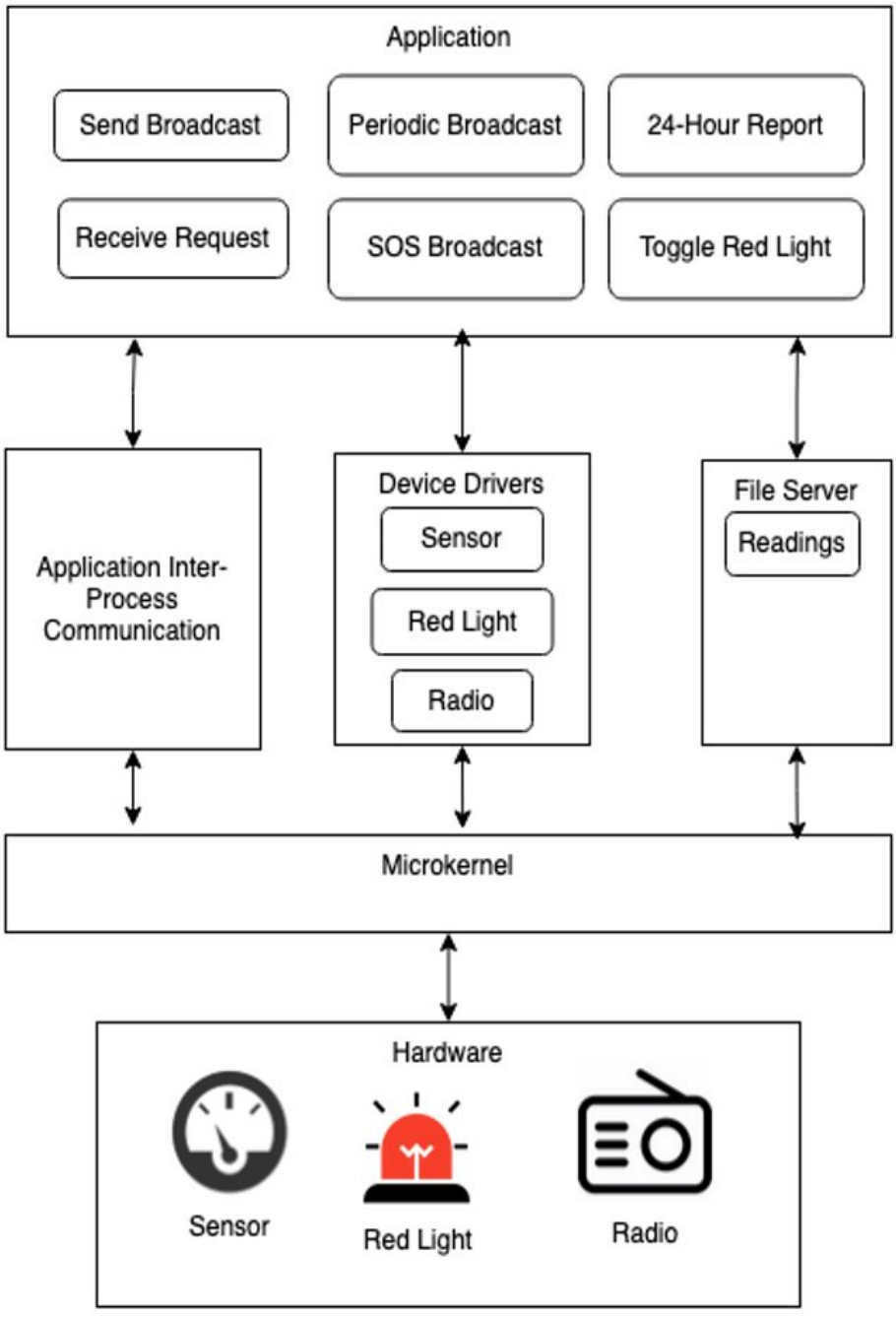
A couple of architectures were also initially considered for the execution architecture. One of which was the layered execution architecture pattern. Here, the system was abstracted into the layers of users, views, processes, peripherals, and external influences. For the processes, they were further divided into some of the main functions that occur at runtime. The separation of layers helps with understanding the flow of the data. Most of the system is focused on the interaction between the processes and peripherals, indicating the real-time embedded nature of the data. The lack of view from the system implies this model is unsuitable for this system's architecture, as it does not show the control of the data regarding orchestration and prioritisation.



2nd Alternative Architecture Pattern – Microkernel

The other alternate architecture considered was the microkernel execution architecture. Here, the kernel contains the minimum services for the system to function. The rest of the Operating System (OS) services are implemented in a separate address space to the kernel. This means some of the OS services, that are in the kernel normally as per a monolithic architecture, are instead separated out and with the rest of the system application program. The main reasoning for this separation is that the system is more loosely coupled because if one of the services fails, the rest of the OS and kernel are not affected. The drawback with this architecture is that the execution speed is reduced, as message passing needs to be established between the client applications and services running the user address space.

With the sea buoy microkernel execution architecture, the device drivers and file server are separate from the microkernel. The application processes can communicate with the services and other processes using the microkernel's application inter-process communication. The necessary hardware is also linked to the microkernel. This architecture highlights the role of the OS services at runtime and loosely couples the system's services. However, it is difficult to follow the flow and control of the data and understand how the processes are linked together. Hence, the microkernel architecture was not initially chosen.



Finalised Execution Architecture and Explanation

For the final execution architecture, the broker execution architecture pattern was combined with the microkernel execution architecture. This gives a more coherent view of the flow and control of the data but still ensures the design is loosely coupled.

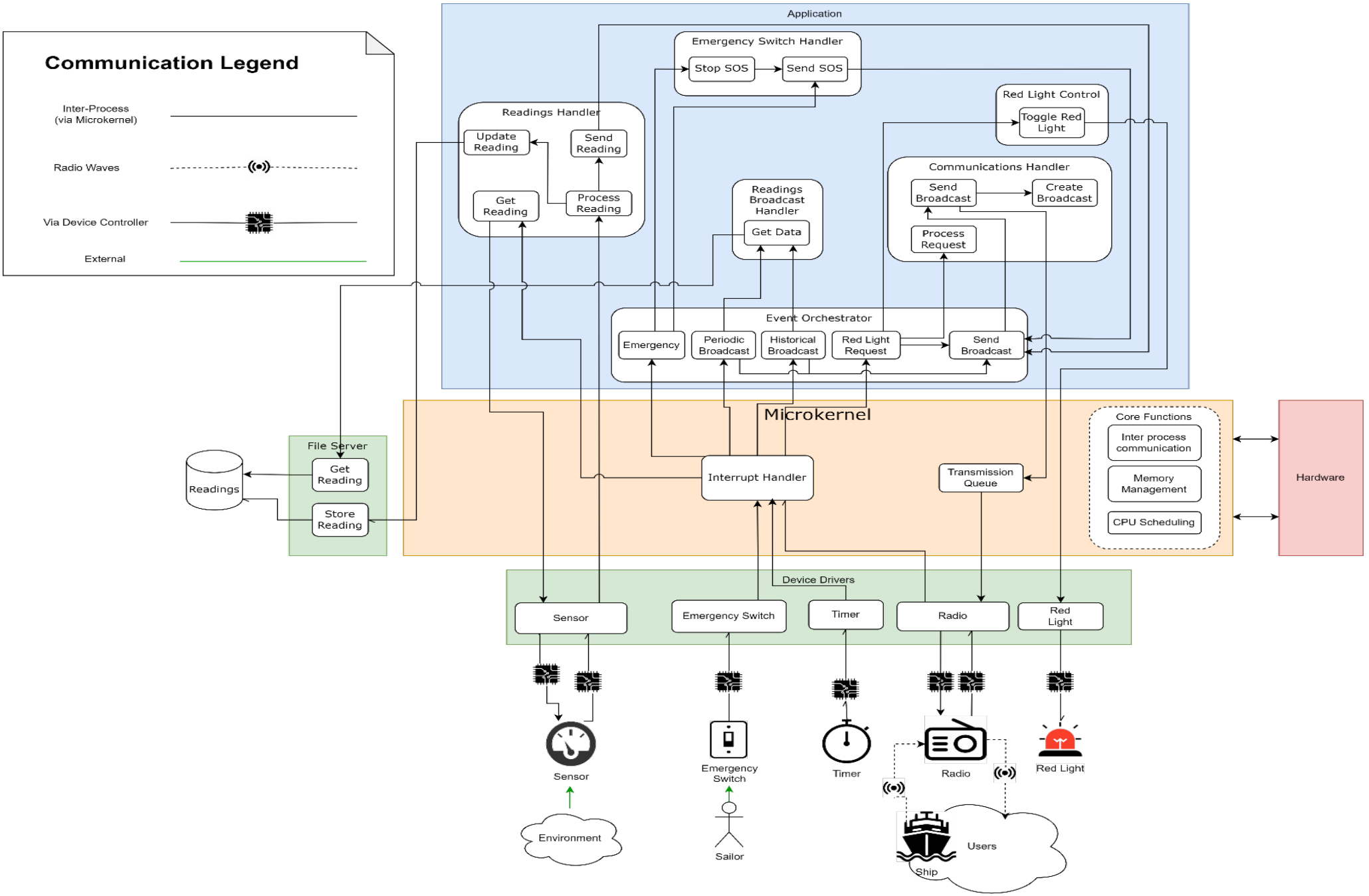
Instead of the broker, this architecture contains an event orchestrator. This was chosen because it has more responsibility than a broker that facilitates communication between client and server entities. It contains more application logic as it handles event-driven actions passed from the interrupt handler.

The microkernel is used for demonstrating the movement of data from the peripherals to the system application. The kernel's core functionalities are listed to indicate the microkernel's purpose. However, the specific kernel processes shown in this architecture (interrupt handler and transmission queue) were chosen as they play a considerable role in this system's runtime. The hardware connected to the microkernel is also included because it shows its proximity to the system's physical hardware.

Using the broker architecture and the microkernel architecture means that the execution architecture is both two-fold loosely coupled. Firstly, the OS services, including the device drivers, are abstracted from the kernel so that if one of the services fails, the rest of the system will not be affected. Secondly, the application components are loosely coupled together with the presence of the event orchestrator. The communications handler is an example of this, as it is only connected to the event orchestrator and no other application component.

The Readings Broadcast Handler was created from the co-location of the periodic and historical broadcast entities in the conceptual architecture. These were closely co-located because they are both broadcasts that retrieve readings from the data store. This close co-location resulted in the readings broadcast handler, whose main purpose is to get the readings data. The periodic and historical broadcast were both kept separate in the event orchestrator since they have different messages for the readings broadcast handler in terms of the time length of readings data they require.

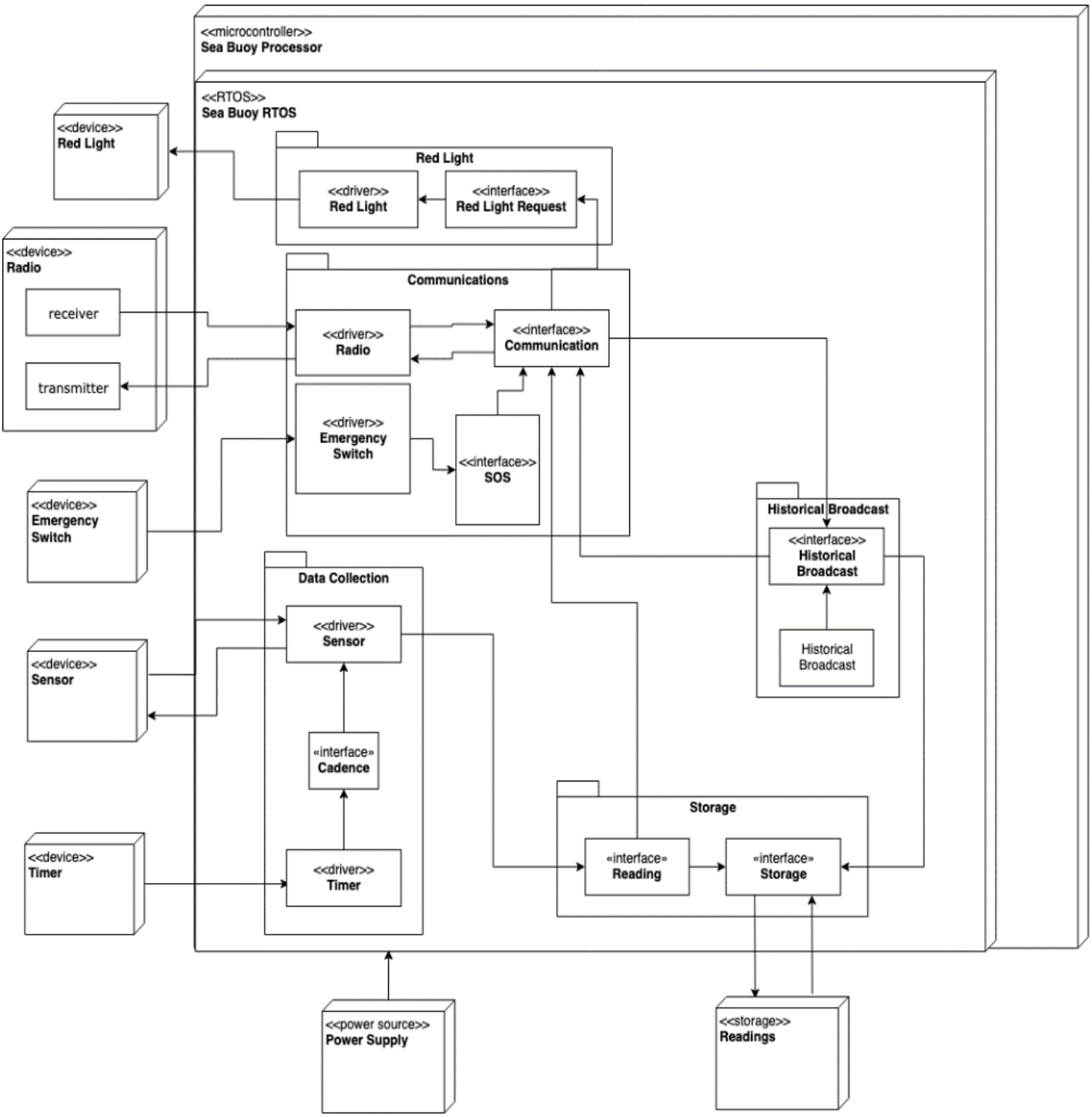
Regarding the communication within the execution architecture, the application processes communicate with each other and the services, via the Microkernel's Inter-Process communication. The communication between the device drivers and the devices is through the device controller. Most of the communications are synchronous as this system is a real-time embedded system, where timing matters.


Communication Legend
Inter-Process (via Microkernel)
Radio Waves
Via Device Controller
External
Application
Emergency Switch Handler
Stop SOS
Send SOS
Red Light Control
Toggle Red Light
Readings Handler
Update Reading
Send Reading
Get Reading
Process Reading
Communications Handler
Send Broadcast
Create Broadcast
Process Request
Readings Broadcast Handler
Get Data
Event Orchestrator
Emergency
Periodic Broadcast
Historical Broadcast
Red Light Request
Send Broadcast
Microkernel
Core Functions
Inter process communication
Memory Management
CPU Scheduling
Hardware
File Server
Get Reading
Store Reading
Readings
Interrupt Handler
Transmission Queue
Device Drivers
Sensor
Emergency Switch
Timer
Radio
Red Light
Sensor
Environment
Emergency Switch
Sailor
Timer
Radio
Red Light
Users
Ship

Implementation Architecture

Initial Implementation Architecture and Explanation

The initial implementation architecture is represented in the following deployment diagram. It consists of peripherals connected to the microcontroller which contains the real-time operating system and application logic distributed as packages. Evaluating against the critical scenario the initial implementation architecture exposes a major flaw. The periodic broadcasts should be initiated by a timer because they are created according to a cadence, in this case 60 seconds. This is not fulfilled in the initial implementation architecture, because instead the periodic broadcast is initiated by the reading interface when the sensor reading has been received. Thus, the initial implementation architecture does not correctly account for the periodic broadcast initiation.



Finalised Implementation Architecture and Explanation

The final implementation architecture adjusted the initial implementation architecture to account for the periodic broadcast mentioned in the Data Analyst user narrative. Here, the timer initiates the periodic broadcast as indicated by its communication to the event orchestrator. Since the timer interacts with the sensor cadences and event handler, it is separated into its own package. Additionally, the historical broadcast package is renamed to readings broadcast package where it is the only package that interacts with the storage package.

Other packages, such as the red light, communications and emergency switch are structured around the main peripheral that they interact with. For implementing the interaction between the devices and application, drivers are included architecture. The microkernel from the execution architecture is not shown since it is included within the RTOS.

Additionally, there were a few packaging cohesion principles that were considered when creating the final implementation architecture.

Cohesion Principle	***How it is applied***
Release reuse equivalency principle	• *Sensor processing will be used for all types of sensors, since sensors are generally similar – all have a cadence, all give readings.*
Common Closure Principle	• *Radio and Communication processing will most likely change together so grouped these together. So, if the radio changes, the communication messages and processing may also change.* • *Changes to data storage may change how readings are processed so these are grouped together.* • *Data collection interacts with communication through the event handler to ensure their separation as they will most likely will not change together.* • *Even though the sensor cadences initiate the readings processing, how the readings are processed should not impact how the sensors are notified, so these should be in separate packages.*
Common reuse	• *Data storage should not be dependent on communication.* • *The red light is isolated since it only interacts with a radio request through the event handler.*

Components	Build/Buy
Micro-controller	Buy
Power source and installation (solar panel and solar battery storage)	Buy
Board support package (support the embedded system. E.g., contains OS image)	Buy
SD-Card for Data Storage	Buy
Application Logic (E.g., Get Reading, Store Reading, Create Broadcast, Process Request, Toggle Red Light)	Build
Peripheral configuration	Build
Data schema (DBMS)	Build

The teams will invest in purchasing the following COTS components: micro-controller, solar panel power source, and a board support package. On the other hand, application logic, peripheral configuration and data structure are components that will be built by the development team. Additionally, the non-functional requirements of reliability, performance, and modifiability were considered for determining these components.

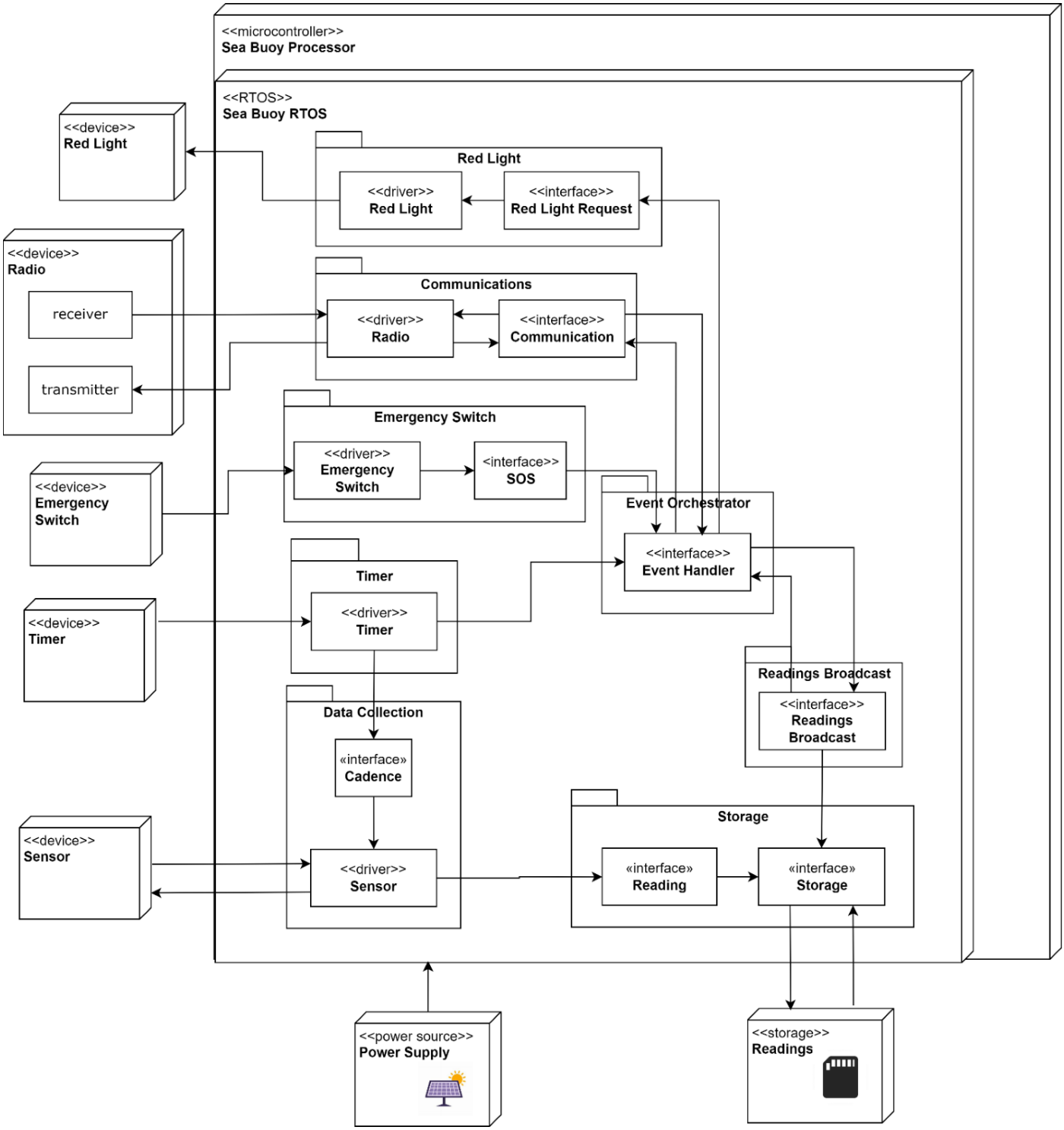
The real-time sea-buoy embedded system will primarily be operated offline; the collected data are stored in the local dataset on the SD-card and retrieved when requested. Initially, implementing the system onto an online real-time system was considered as the cloud enables backup storage and easy access management. However, due to the environmental constraint, the isolation of sea buoys means internet connection is limited or non-existent which greatly affects the data transmission required for an online real-time system. Thus, the implementation solution is defined as a real-time embedded system with the absence of the internet communication.

With a real-time embedded system, there are two main implementation options: a microprocessor and a microcontroller. The microprocessor does not typically come with memory but has more processing power when compared to a microcontroller. The microcontroller, though smaller in size, typically comes with memory and contains more peripherals on a chip. This makes it easier for hosting the software and OS, as well as connecting the different devices to it. Since this is a relatively simple system isolated to each sea buoy, the microcontroller should have sufficient processing power to execute the system logic. Thus, the microcontroller was chosen over a microprocessor for the sea buoy's real-time embedded system.

Additionally, solar panels and its back-up battery storage will be purchased, to serve as the power source for the microcontroller on the sea-buoy. In the circumstance where there are consecutive rainy days with no solar energy available, the solar back-up (battery) is activated to keep the computer operating. Both the solar panel and its back-up battery storage are covered under the power source component in the implementation architecture diagram.

The board support package (BSP) consists of essential elements and programme that help the development teams during the system implementation. This means that the software components related to hardware integration do not have to be programmed from scratch. Using the initial programme and/or elements in the BSP, the system application logic and peripheral configuration can easily be built and integrated into the microcontroller.

The microcontroller's memory will be used to store the RTOS and an SD-card will be used to store the readings. Since the readings are taken in tens of seconds, the speed of the SD-card should be adequate. Additionally, SD-cards are more affordable compared to other options such as SSD or Hard Drives and require little power. The SD-card should have adequate storage space since the data stored in the database will be cleared out and refreshed every 24-hours to prevent the system from corruption caused by excessive storage. SQL will be utilised as the system data schema due to its' simplicity and effective data storing and retrieval is optimal for the system operation.


<<microcontroller>>
Sea Buoy Processor
<<RTOS>>
Sea Buoy RTOS
<<device>>
Red Light
Red Light
<<driver>>
Red Light
<<interface>>
Red Light Request
<<device>>
Radio
receiver
transmitter
Communications
<<driver>>
Radio
<<interface>>
Communication
Emergency Switch
<<driver>>
Emergency Switch
<interface>>
SOS
<<device>>
Emergency Switch
Event Orchestrator
<<interface>>
Event Handler
Timer
<<driver>>
Timer
<<device>>
Timer
Readings Broadcast
<<interface>>
Readings Broadcast
Data Collection
«interface»
Cadence
Storage
<<device>>
Sensor
<<driver>>
Sensor
«interface»
Reading
«interface»
Storage
<<power source>>
Power Supply
<<storage>>
Readings

Rationale

When designing the overall system, different architectural patterns were considered. This resulted in elements of the architecture patterns being incorporated into the design.

Broker Architecture

With the broker architecture, the broker is primarily responsible for coordinating the communication between different clients with the right servers. The broker architecture possesses loose coupling, which was a feature that was desirable for the system. Additionally, the broker architecture forms a suitable basis for event-driven architecture as it can handle multiple client requests.

Taking this into account, a broker was initially included in the design, however it required more application logic than just facilitating communication between clients and servers. The broker provided the foundation for creating the event orchestrator, as it still received requests from the clients, where the clients were the messages passed from the interrupt handler. Designing from the broker initially, allowed the architecture to maintain its loose coupling for reliability and modifiability.

Below is a list of advantages and disadvantages of the broker architecture.

Advantages	***Explanation***
Openness	*Flexibility of using hardware and software of different vendors*
Concurrency	*Concurrent processing leads to an increase in performance*
Scalability	*It can increase the throughput by adding new resources*
Fault tolerance	*The ability to continue in operation after a fault has occurred*

Disadvantages	***Explanation***
Complexity	*More complex than centralized systems*
Security	*It is more susceptible to external attacks*
Manageability	*More effort required for system management*
Unpredictability	*Unpredictable responses depending on the system organization and network load.*

Monolithic Layered

The Monolithic Layered architecture highlighted the importance of the Real-Time Operating System for the Sea Buoy real-time embedded system, as the initial execution architectures did not include OS functionalities. The benefit of this architecture pattern is that less software is involved so the performance is enhanced. However, its tightly coupled design was unsuitable for high reliability, hence this architectural pattern was rejected.

Below are the list of advantages and disadvantages of the Monolithic layered architecture:

Advantages	***Explanation***
Modularity	*Each layer only executes the tasks it is scheduled to perform and hence modularity is supported in this design*
Abstraction	*As each layer has its own set of functions, the implementations of other layers are abstract to it*

Disadvantages	Explanation
Scalability is limited	*It uses tight coupling design principle and as the system increases in size, it will have to be redesign into another architecture*
Performance	*Speed is inhibited by the separation of the layers as requests must transverse the other layer before execution is achieved.*
High cohesion and tight coupling	*The design principles do not meet the requirement of the system to be loosely coupled. Real-time system with high cohesion and tight coupling increases system inability to function when one component is disrupted*

Microkernel

Features of the Microkernel architecture were used in the final execution architecture. The loose coupling with the Operating System means that the reliability of the system could be applied to the Real-Time Operating System. The drawback of the microkernel was that the abstraction meant that it could possibly be slower when compared to the monolithic layered architecture. However, when considering the stakeholder non-functional requirements, reliability and modifiability were prioritised higher than performance since it is believed the performance requirements can still be met using the microkernel.

Below is a list of advantages and disadvantages of the micro-kernel:

Advantages	Elaboration of the advantages
Fault tolerance	*Services can continue operation despite having a failure due to the isolation of the services*
Performance	*Compact and isolated hence performance is increased*
Modifiability	*Modular and various modules may be swapped, reloaded, and modified without affecting the kernel*
Security	*Less susceptible to attacks due to the isolation*

Disadvantages	Elaboration of the disadvantages
Cost	*Providing services are more costly than in a traditional monolithic system*
Efficiency	*If the services or the functions are implemented as procedures or processes, code-switching or functional call is needed between the services.*
API	*The API specifications need to be checked rigorously in advance as each services interact using public API.*
Database	*Enabling transactions related to multiple microservices would be formidable.*

Evaluation

System Qualities

Quality	Description	Metric
Availability (Reliability)	The system should be able to operate and be available without any outages	Broadcast breakages, lack of broadcasts
Performance	The system should be able to handle the timing constraints and priorities	The broadcasts are all on time, no later than five seconds. The SOS signal should be transmitted before the 24-hour broadcast, and the 24-hour broadcast should be transmitted before the periodic broadcast.
Scalability	The system should be able to support peak message traffic with devices (broadcasts, readings, requests).	The system should be volume and stress tested to determine its limits and that it can handle at least 1000 messages.
Modifiability	The system should be able to support changes to the sensors and the priorities.	Modifications only require one maintenance trip.
Data Integrity	The system should preserve the accuracy of readings and ensure that no data is lost or corrupted during transmissions.	Received periodic broadcasts match the sensor readings. Use checksums to ensure data remains unchanged.
Testability	The system should run a test on all components to ensure its' functionality	During the maintenance process, specific function test is conducted to ensure that the system functionality is still

Among all the system qualities, the above-mentioned qualities were identified as crucial for the team to focus on to fulfil both the sea-buoy requirements and enhance the system functionality. The stakeholders and their role in relation to the system are taken into consideration when evaluating these qualities.

Alternatively, usability is the quality that was not specified, as it is less crucial where user and system communication occurs over sea buoys and radio. The team also manages the system constraint of real-time systems having limited user interfaces.

The identified qualities are further analysed using the ATAM analysis significant scenario below, where each quality is compared against others and evaluated how they would impact the system design principle.

The following scenarios were considered when evaluating the candidate architectures. These scenarios were created based on the competing non-functional requirements and the identified system qualities.

Sea-buoy system valued and prioritised its performance qualities where the response and processing time are restricted within a time frame. Most of the qualities are pitched against performance as it is the key considerations.

Scenario 1: Reliability vs Performance

The system is required to be highly reliable since it is inconvenient to perform regular maintenance. It also needs to have considerable performance as this is a real-time embedded system where timeliness is crucial as it frequently transmits real-time data and responds to real-time requests. Thus, loose coupling is conducted to achieve high reliability. We increase the availability by limiting the dependencies to completely minimize system failure during its operation. However, this will also decrease performance as there is less direct communication between components.

Scenario 2: Modifiability vs Performance

The system needs to be easily modifiable so it can be compatible with different system requirements, such as adjusting the number of sensors or changing the priorities and cadences. Modifiability to add new tasks and rearrange its priority helps the system determine which action to perform first from a series of requests received.

For instance, if two passing vessels simultaneously request, with one requesting for reading data and the other for broadcast the SOS signal. In this instance, the system would transmit the data after broadcasting the SOS signal as the second task priority is higher. The system is more effective and efficient as a result of the capability to be modified by the user. However, the low communication between components in a loose coupling design will have an impact on the system performance in the future when additional functions are added to the system where task execution may overlap.

Scenario 3: Scalability vs Performance

The system will be implemented on numerous sea-buoys and acquire greater processing power is necessary to improve its' performance. Nevertheless, the budget constraint will prevent the teams from solely investing to increasing the system processing power to be able to cope with significant tasks are executed concurrently. Consequently, increasing scalability is equivalent to decreasing in performance, as more data are being handled.

Scenario 4: Portability vs Performance

The system must be compatible with different sea-buoy types and their sensors. So, loosely coupled design enable components to be easily added/removed without compromising other functionality in the system. This is a great advantage, but as discussed other scenario, this design also has minor disadvantages to the system's performance.

Scenario 5: Reliability vs Modifiability

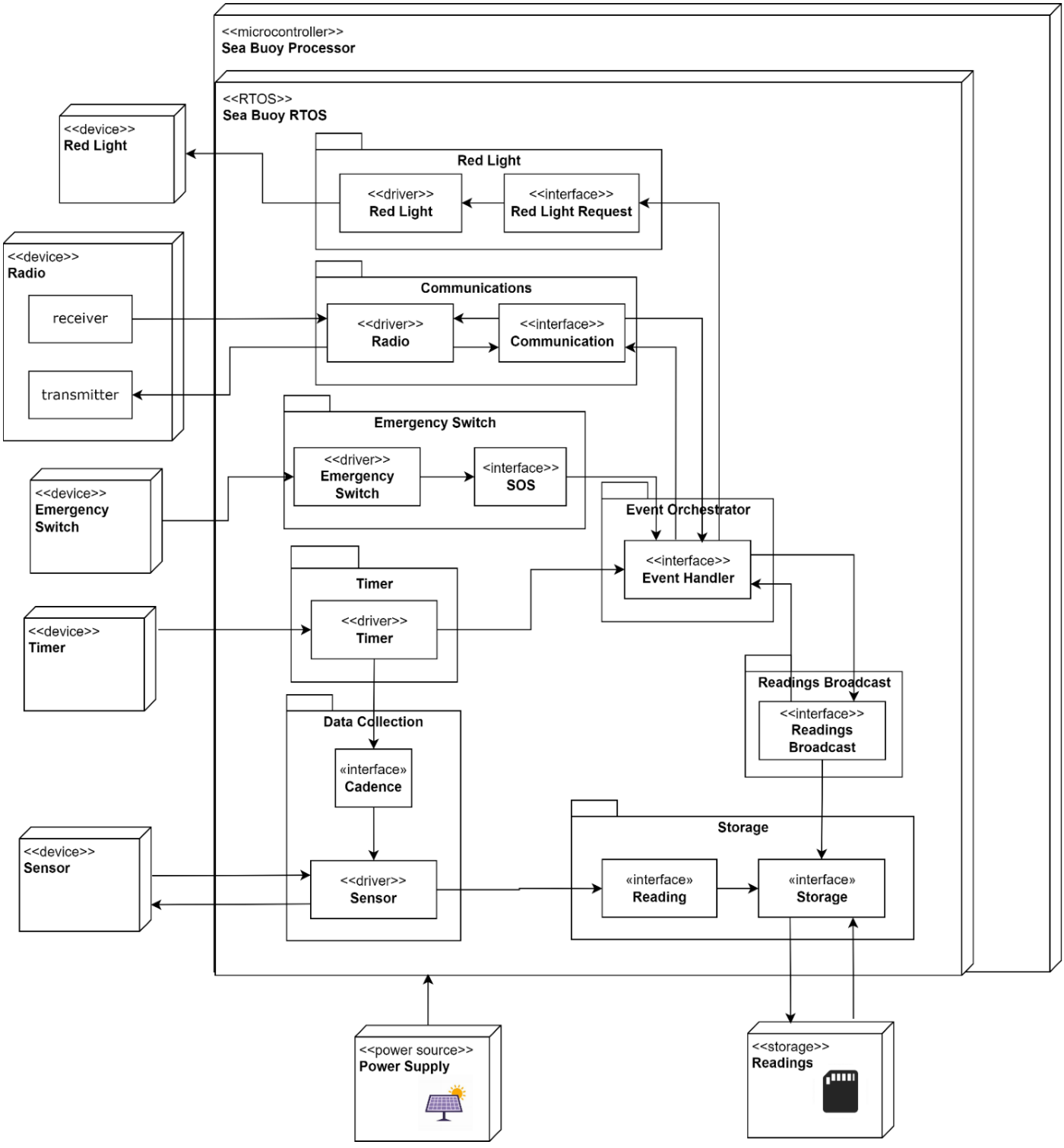
Changes to existing parts of the system may need to be modified, such as the current broadcast priorities and sensor cadences. Care should be taken that the system downtime is kept at a minimum during modification, and that the system can remain reliable after updates.

Scenario 6: Scalability vs Data Integrity

The system needs to ensure that readings remain accurate, and no data is lost or corrupted when receiving or transmitting data from the sea buoy. The software should use checksum in a large scale of data checking and the system make sure that the received broadcast match the sensor readings. Dramatic increases on message traffic, may make keeping this data integrity difficult.

Scenario 7: Scalability vs Testability

The maintenance of the system should run a test on how functional the update of the software runs, and each component should work. When scaling the software and applying it to multiple sea buoys, it may become more difficult to test as the scope of the system increases as well as the number of variances between sea buoys.



Scenario 1: Reliability vs Performance

The architecture uses an event orchestrator to create a loosely coupled design to achieve high reliability. If one component fails, the rest of the system will not break (unless it is the event orchestrator). However, using the event orchestrator also means that there is another body of communication for messages to go through, meaning that the speed of the message passing is slower, compared to if the entities were directly communicating with each other. To compensate for this, the sensor's reading is directly passed from the sensor driver to the reading interface, so it can be quickly processed and stored in the SD-Card.

Scenario 2: Modifiability vs Performance

The architecture's packaging structure makes it easy to modify different components, without having to change the whole system. For example, adding/removing sensors only affects the data collection area. The event orchestrator allows changes to priorities to be made, without having to change the communications package.

Scenario 3: Scalability vs Performance

Using a microcontroller is more affordable than a microprocessor. This means that the performance capacity of the sea buoy is less when compared to a microprocessor. However, since this is a simple system that is isolated on each sea buoy, the microcontroller should adequately be able to handle the performance requirements.

Scenario 4: Portability vs Performance

Once again, the loosely coupled design makes it easy to implement the optionality of certain features (E.g. red light, historical broadcast). This enables the system's most basic format to simply collect and transmit readings using the data collection, storage, event orchestrator, communications packages, and their respective peripherals.

Scenario 5: Reliability vs Modifiability

The loosely coupled design means that modifications to the components can be executed without affecting other parts of the system. This means reliability will not be affected when changes are made to parts of the system. For example, modifying the sensors is isolated to the data collection area, meaning the system will still be able to handle other components such as communications and emergency broadcasts.

Scenario 6: Scalability vs Data Integrity

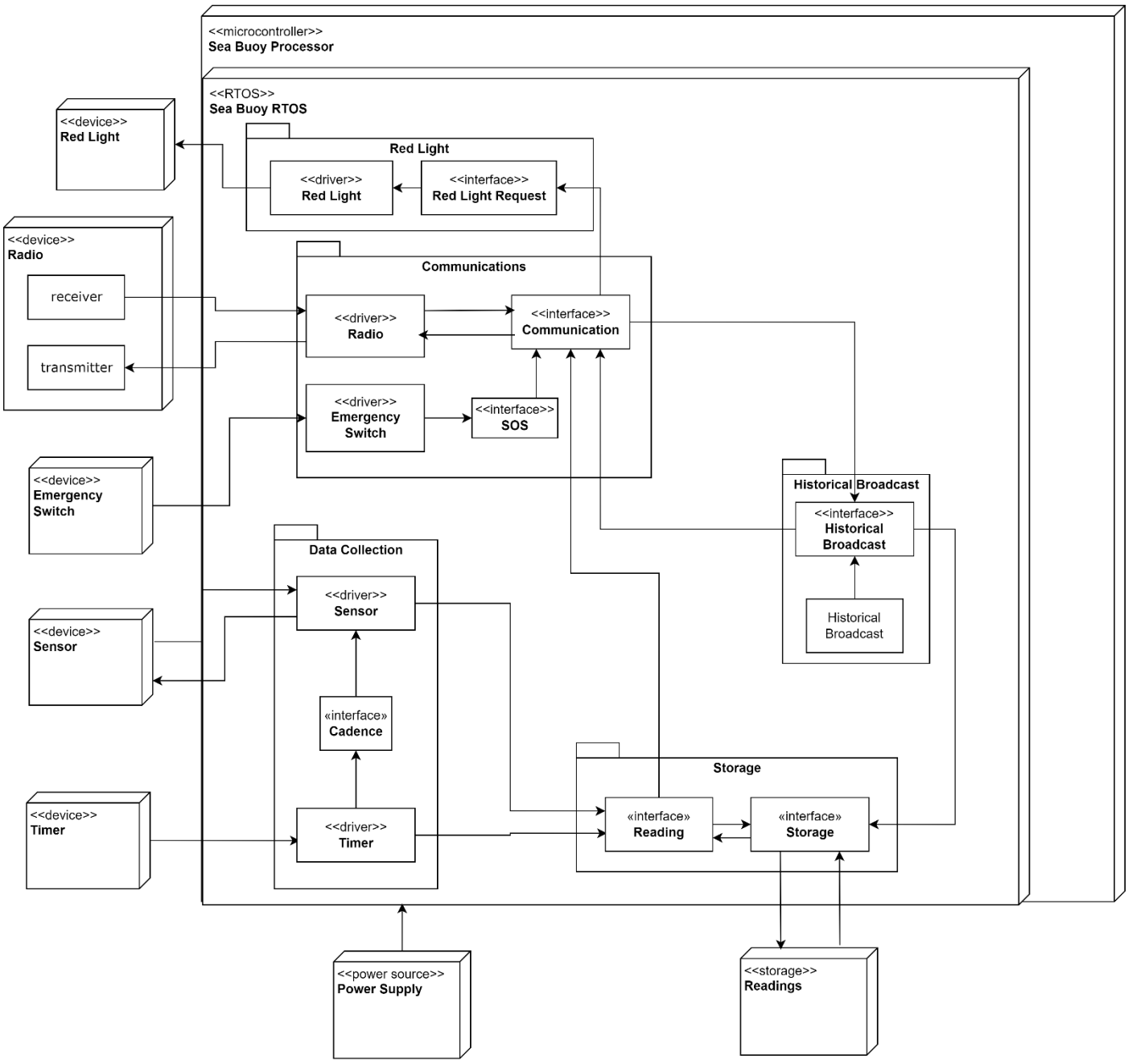
The package structure of the architecture makes it easier to check whether the received period broadcast was matching the sensor reading data inside SD-card. This makes it easier to make sure that it the readings to correct and not corrupted or lost during transmission from storage to SD-Card. Having the storage interface as the only main point of distributing readings, also means that data integrity can be handled in one place.

Scenario 7: Scalability vs Testability

The design of loosely coupled makes it easier to test certain features (E.g., red light, emergency switch and sensor). This makes the system simple to test all the feature so that the reading and each feature functionality remain stable. When the system is easier to test, it becomes easier to scale as the system quality standards can be met quicker.

Alternative Architecture Evaluation with ATAM Analysis

This alternative architecture still uses the microcontroller with the real-time operating system and connected devices. The packaging structure differs, however, as this alternative architecture is more optimised for performance. It does this by removing the event orchestrator and providing some entities with direct communication. For example, the periodic broadcast, created from the reading interface is passed directly to the communications interface without having to pass through the event orchestrator. Additionally, the packaging is restructured to group more responsibilities together for faster communication. This includes the timer driver being in the data collection area and the emergency switch driver and SOS interface located in the communication package.



Scenario 1: Reliability vs Performance

In this architecture, the components are still somewhat loosely coupled together mostly via the communications interface. However, since the communications interface holds more of the event responsibility, the communications interface decreases the reliability of the system as it has greater dependencies and responsibilities. Although, connecting the components directly to the communications interface means that messages are passed directly between entities. Thus, this system's performance is not drastically reduced.

With the differing packaging structure, more responsibilities lie within the communications package, which decreases the reliability of the system since there are more dependencies. However, placing the emergency functions with the rest of the communications handling increases the system's performance for handling emergencies.

Scenario 2: Modifiability vs Performance

This architecture caters to some modifiability. For example, adding/removing sensors or changing their timings would only affect the data collection area, as they logically should. However, any changes to the communications package may affect majority of the system, making the system harder to modify.

Scenario 3: Scalability vs Performance

Using a microcontroller is more affordable than a microprocessor. This means that the performance capacity of the sea buoy is less when compared to a microprocessor. However, since this is a simple system that is isolated on each sea buoy, the microcontroller should adequately be able to handle the performance requirements.

Scenario 4: Portability vs Performance

Once again, the loosely coupled design makes it easy to implement the optionality of certain features (E.g., red light, historical broadcast). This enables the system's most basic format to simply collect and transmit readings using the data collection, storage, communications package, and their respective peripherals. This is simpler compared to the other architecture which has the event orchestrator.

Scenario 5: Reliability vs Modifiability

The separation of the components into packages does facilitates some modifiability. For instance, changes to the sensors will only affect the data collection area. However, with more responsibilities grouped in packages, it means that not as much of the system can function if modifications are being conducted. This can be highlighted if changes were made to the emergency switch, where this may affect the system's communication component, thus decreasing the system's reliability.

Scenario 6: Scalability vs Security

Having the reading interface send readings for the periodic broadcast as well as the storage interface send readings for the historical broadcast, means that it an influx of readings sent can be handled. However, with these two packages sending readings, it exposes more points of failure for data corruption and loss. Additionally, having the communication interface directly receive readings as well as radio requests, also makes it easier for data to be mixed up, thus impacting the data integrity.

Scenario 7: Scalability vs Testability

The smaller number of packages means there are more responsibilities within some of the packages. This makes it difficult to test and debug each package. Difficulty in testing can result in difficulty in scaling the system, because it delays meeting the quality assurance standards.

Evaluation Summary

The following summaries the quality trade-offs between these two candidate architectures.

Final Implementation Architecture	***Alternative Implementation Architecture***
Better Reliability, Modifiability	*Worser Reliability, Modifiability*
Worser Performance	*Better Performance*

As highlighted above, the main trade-off point is between the system's performance versus reliability and modifiability. The system's performance demands are a concern as it has real-time requirements. However, considering the timings are in tens of seconds and not in milliseconds, the performance demands are extreme, making this a slightly soft real-time embedded system.

Here, both architectures would be able to satisfy the performance requirements reasonably. Additionally, regarding the qualities that stakeholders deem important, reliability and modifiability are essential, considering the need for infrequent maintenance. Since the final implementation architecture possesses better reliability and modifiability, it is chosen over the alternative implementation architecture.

Conclusion

In conclusion, the user stories listed were based on the primary stakeholders and covered every aspect of the system, meeting the requirements of the case study presented. The user narratives added additional insights into each stakeholder's perspective based on the user stories. Risks and constraints and their mitigation were carefully thought out and evaluated. Before moving on to the formulation of the various architectural patterns, assumptions about the system were made.

The conceptual architecture started off as a simplified, abstract ideology of context diagram providing an overview of how the system was to function to achieve its objectives. However, as the team explored deeper into each aspect of the system and how the components interact with each other, a more complex and sophisticated architecture emerged, which became the final conceptual architecture.

The execution architecture combined the two alternative architectures together to achieve the most desirable outcome for this system. Having the benefit of isolation among many other advantages of the microkernel, the system would be able to function even if a fault occurs, allowing the system to be more reliable and robust. The performance is enhanced by the usage of the event orchestrator, which is the upgraded version of a broker.

The final implementation architecture is well-designed and refined. It ensures loose coupling to achieve high reliability. Different aspects considered were covered and explained in-depth, comparing, and displaying the thought process that accompanied the creation of the architecture.

In summation, through the peer review feedback received and additional research conducted, all architecture created were carefully selected and in-depth explanations were included. Additionally, alternative architectures were provided, which exhibit the extensive research and consideration the team had undergone to produce the optimal architecture for the sea buoy system.

Team Contribution Table

Meetings are conducted on MS Teams and the group will discuss and complete the required task and prepare for the peer review presentations. All teams' members have successfully completed their assigned work within the set due date for each task. Group 9 has organised meetings on-campus (before tutorial class 13th, 20th and 27th October) to discuss and complete the execution, implementation architecture and system evaluation.

Contributor	Contribution(s): (Task/Sections)	Score *1-min. / 5-max.*
Tamsin Low, 13925969	System Purpose and Objectives Assumptions Stakeholders User Needs/Narrative Constraints Conceptual Architecture Execution Architecture Implementation Architecture Rationale Evaluation	
Wei Ming Edward Ong, 14005817	Report summary Assumptions User stories Execution Architecture Implementation Architecture Alternative Architecture Evaluation Diagram Conclusion System Qualities Explanation Significant Scenario Explanation I participated in the weekly contribution which was essentially the initial phase for the report. When the team started on the final report, I rendered my help by contributing to the above-mentioned sections along with the transference of information which we have collated over the past weeks.	
My Duong, 13934070	Report summary System purpose and objectives Assumptions User Stories Functional and non-functional summary Conceptual Architecture Execution Architecture COTS components Significant scenario I participated in completing each section throughout the peer review presentation weeks. Hence, I started transforming the information from the presentation that the group has completed into this report from 27th October and finalise the report on 30th October before submitting the assessment.	
Sakura Adachi, 24648418	User stories Execution Architecture Alternatives Explanation Implementation Architecture Explanation Implementation Architecture principles I participated in this report by working on the section above. Every time we had peer review, I started planning on what technology tool we should utilize as well as the system architecture we should use for this assignment, and had focused on writing on the report from the 28th to 30th of November. Additionally, I put my efforts into contributing to a presentation by researching the technology we used for the week. I gathered the primal information and knowledge for the report during that time.	

Andrew Soen, 14189043	User Stories Conceptual Architecture Finalised Implementation Architecture Significant scenario I contributed to the assigned work each week during peer reviews and contributed some section listed above on this report.	
Nexus Baquir, 13572324	Beginning the peer reviews, I contributed to the assigned work weekly. Thus, in this report adding to the following sections: Stakeholders User stories and narratives Risk and mitigation Constraints Conceptual entities COTS Components Proof reading and grammar correction	