UNIVERSIDADE D
COIMBRA

José Diogo Terêncio Sobrinho

AUTOMATIC USER INTERFACE GENERATION FOR MULTI-USER/MULTI-DEVICE (MPCS)

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Jorge Cardoso and Professor Licínio Roque and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September, 2023

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

José Diogo Terêncio Sobrinho

Automatic User Interface Generation for multi-user/multi-device (MPCS)

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Jorge Cardoso and Professor Licínio Roque and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September, 2023

Acknowledgements

I would like to thank my family and friends...

Funding

This work was partially funded by MPCS project #101048546 under UCPM-2021-PP-MARIPOL, programme UCPM2027 and by the FCT - Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit - UIDB/00326/2020 or project code UIDP/00326/2020.

Abstract

The combat against marine pollution must be given special attention. In Portugal, conducting a simulation to train the appropriate entities to fight pollution is very expensive and time-consuming. To counteract this, a game was developed for learning purposes called Marine Pollution Control Simulator (MPCS), which simulates maritime pollution events, more specifically, oil spills, where the future users can train so that they can be more effective in combating real maritime pollution events, without spending large amounts of money, and doing it much faster than normal. The implementation of this project was divided into five different dissertations, each having a part in the development process of the project.

When the initial ideas for the project were discussed, a problem was encountered: the diverse number of possible interfaces in the game, given the diversification of entities taking part in the game, can significantly increase the developing time. That said, there was a need to reduce development time. With this in mind, this dissertation demonstrates a model for automatically generating interfaces using templates capable of reducing the development effort, implemented when developing the game. Using this model made it possible to develop the game much faster than creating all the interfaces manually. However, this model had some problems. That said, this dissertation also proposes another automatic generation model using a JSON configuration file, capable of solving the generation problems using templates, such as the customization of interfaces and some aspects of abstraction that needed to be met.

User studies were carried out to evaluate the interfaces generated. With the analysis of the results, we could conclude that the generation of the interfaces was a success, with only a few aspects to improve.

Ultimately, we can conclude that the objectives of this dissertation have been met. However, there is still room to improve the models developed to create a solution that can better meet the client's needs and make the game experience more enjoyable.

Keywords

Automatic User Interface Generation, Model-based User Interface Generation, Marine pollution Control Simulator, User Interface Language Description

Resumo

O combate à poluição marinha deve merecer uma atenção especial. Em Portugal, a realização de um ensaio para formar as entidades competentes no combate à poluição é muito dispendiosa e demorada. Para contrariar este facto, foi desenvolvido um jogo com fins didáticos chamado Marine Polution Control Simulator (MPCS), que simula eventos de poluição marítima, mais concretamente derrames de hidrocarbonetos, onde os futuros utilizadores poderão treinar para serem mais eficazes no combate a eventos reais de poluição marítima, sem despender grandes quantias de dinheiro, e fazendo-o muito mais rapidamente do que o normal. A implementação deste projeto foi dividida em cinco dissertações diferentes, cada uma com uma parte no processo de desenvolvimento do projeto.

Quando as ideias iniciais para o projeto foram discutidas, foi encontrado um problema: o número diversificado de interfaces possíveis no jogo, dada a diversificação de entidades presentes no jogo, pode aumentar significativamente o tempo de desenvolvimento. Assim sendo, houve a necessidade de reduzir o tempo de desenvolvimento. Neste sentido, esta dissertação demonstra um modelo de geração automática de interfaces através de templates, capaz de reduzir o esforço de desenvolvimento, implementado no desenvolvimento do jogo. Com a utilização deste modelo foi possível desenvolver o jogo muito mais rapidamente do que criar todas as interfaces manualmente. No entanto, este modelo apresenta alguns problemas. Dito isso, esta dissertação também propõe um outro modelo de geração automática usando um ficheiro de configuração em JSON, capaz de resolver os problemas de geração usando templates, como a customização das interfaces e alguns aspectos de abstração que precisavam de existir

Foram realizados testes de usabilidade para avaliar as interfaces geradas. Com a análise dos resultados, foi possível concluir que a geração das interfaces foi um sucesso, com apenas alguns aspectos a melhorar.

Em última análise, podemos concluir que os objetivos desta dissertação foram cumpridos. No entanto, ainda há espaço para melhorar os modelos desenvolvidos para criar uma solução que possa responder melhor às necessidades do cliente, e que possa tornar a experiência de jogo mais agradável.

Palavras-Chave

Geração Automática de Interfaces de Utilizador, Geração de Interfaces de Utilizador baseada em modelos, Simulador de Controlo da Poluição Marítima, Linguagens de Descrição de Interfaces de Utilizador

Contents

Acronyms

AUI Abstract User Interface.

AUIG Automatic User Interface Generation.

AUIGM Automatic User Interface Generation Methods.

CCC Command and Control Center.

CUI ConcreteUser Interface.

DCPM Directorate for Combating Pollution of the Sea.

DEI Department of Informatics Engineering.

DGAM Central Services of the Directorate General of Maritime Authority.

GBL Game-Based Learning.

GUI Graphical User Interface.

HC HydroCarbons.

HP Health Points.

MAS Maritime Authority System.

MPCS Marine Polution Control Simulator.

NMA National Maritime Authority.

PUC Personal Universal Controller.

UI User Interface.

UIDL User Interface Description Language.

VR Virtual Reality.

List of Figures

List of Tables

Chapter 1

Introduction

In Portugal, the Central Services of the Directorate General of Maritime Authority (DGAM) and the Directorate for Combating Pollution of the Sea (DCPM) are aware of sea pollution events from all possible sources, hence being in charge of Marine Pollution Control. The objective of Marine Pollution Control, as for hydrocarbon (oil) spills, is the containment, collection or dispersion, the cleanup of the contaminated areas on the shore and their storage or disposal as quickly as possible and, of course, at the lowest cost [1]. To learn how to combat these events, they need to have experience and training, so they are prepared and execute the best aid possible when the time comes. Regular practical training is required to keep the Marine pollution Control in a high state of readiness, such as:

- Coordination of the means involved (direction, actions, sequence and timeliness of actions, information management, means management, security, and effectiveness).
- Communication (information, orders, clarity).
- Procedures (individual and teams): Logistics (movement of means), Technicians (equipment operation), Legal, national, and international administrative (appointments, information, request for support).
- Equipment Maintenance.

The training performed currently is not at all the best practice because it has a high cost of execution, it cuts the traffic in the area where they are doing it and and it takes a long time to do it, although they do it in a controlled area. Because of all these costs, they cannot often perform this training and consequently cannot reach the maximum effectiveness in combating the problem.

1.1 Motivation

As explained earlier, the Marine Pollution Control entities must train and simulate the combat to the pollution events to be ready when the time comes. Still, the conduct of this training is expensive.

To solve this problem, the department of Informatics Engineer of the Faculty of Science and Technology of the University of Coimbra in association with Autoridade Marítima Nacional, EVM, IPTL, and Qualiseg are working together to develop a software to simulate real-life scenarios and train to combat them, called the Marine Polution Control Simulator (MPCS). MPCS main goal is to develop a cloud-based platform easily reachable through different platforms (mobile, tablet, or laptop) that allows the Game-Based Learning and performance evaluation of marine pollution control operations and coordination.

MPCS is a tool for Game-Based Learning (GBL) that allows [1]:

- Skills development of first-responder teams and management teams under complex multiplayer scenarios based on a discrete event simulation methodology.
- Exercising scenario-based approach that aims towards individual and collective experience and improvement of operational coordination in Marine Pollution Control scenarios.
- Performance self-assessment by participants (executed versus expected/reference).
- Development of Collective Competence using state-of-the-art virtual environments.
- Response Capacity Assessment of the actors involved in an incident-based scenario.

The platform will be implemented in multiple devices, for multiple users, and it will feature a considerable number of different persons, equipment, material, elements, each one with a different interface. The user will be able to experience real-life scenarios, such as, for example, driving a car, placing barriers in the containment area so that the spill does not spread, interacting with other users, or using skimmers to recover spilled oil. The User Interface (UI) should be able to adapt to every context and environment and every user's preference of use. The UI is essential because it needs to meet user expectations and support the effective functioning of our application.

The main problem this dissertation aims to address is the wide variety of user interfaces in the simulator, each one for every interaction possible. With that said, there is a need to use Automatic User Interface Generation Methods (AUIGM) that, depending on the interaction's available information, will create the most suitable interface. Using these methods also can bring other advantages to the implementation since creating all these interfaces manually and saving them in the system can be inefficient. It would waste unnecessary use of resources and

extra developing effort, provoking some issues, such as massive code replication and maintenance issues. Replication of code happens because the implementation for each interface is similar, changing only a few aspects between them. The maintenance issues occur because, as it isn't a modular solution, a change to the interface code, whether for fixing a problem or a new feature, would require a change in every interface created.

AUIGM are beneficial to fix these issues. Using these methods, we only need to implement the necessary code to manage every type of device or user and create the interfaces for each. The interfaces are not created statically but depend on the context of use. In other words, the interfaces are context-sensitive and generated in real time. These methods would solve the replication of code, maintenance issues, and design effort, as creating specific interfaces for every type of context is unnecessary. Furthermore, they would save a good amount of resources, as the interfaces are not pre-saved in the system but are created at the moment of use.

Therefore, this dissertation will study AUIGM and report their implementation in MPCS.

1.2 Scope

This dissertation reports the study of AUIGM and their implementation in the MPCS project. Furthermore, a game was developed for learning and evaluation purposes, where the users can simulate real-life scenarios of pollution events, specifically oil spills, and study and evaluate means to combat it.

The development of the MPCS project is divided into five different components, more specifically, (1) a digital twin where the goal is to model and simulate real-world phenomena (actors, organizational structures, environments, actions, and resources), (2) a game editor and simulation model, that will take care of implementing a model that calculates in real-time the responses to the actions of the various agents in the simulation, and is in charge of creating simulation scenarios, (3) the Virtual Reality (VR) operations, where it's concerned to implement a virtual reality interface for some actions in the simulator, (4) the multiplayer server architecture that defines the overall multiplayer service architecture of the MPCS simulation game system and finally (5) the UI generation, which is the one this dissertation will report.

This dissertation's project aims to develop every UI present in the simulator by using AUIGM. Since there will be several types of interactions in the simulator, the presence of multiple UIs creates a need to study and implement AUIGM to spare design effort and other problems.

1.3 Objectives

This work aims to study and analyze existing automatic user interface methods and implement a solution using these methods in the context of the MPCS project. More specifically:

- Define the solution's architecture and explain how the automatic user interface generation methods solved the problem.
- Develop a reference implementation for programmers wishing to use the solution in the used MPCS project.

1.4 Document structure

Chapter 2 details the context of the practices conducted by the Marine pollution control entities, such as their training, combat procedures, and our vision for what the simulator should be.

Chapter 3 presents some essential concepts necessary for understanding the Automatic User Interface Generation (AUIG). It conducts a study on User Interface Description Language (UIDL)s and approaches that utilize AUIGM. A comparison is made, and a review of the acquired knowledge is provided.

Chapter 4 outlines the work plan devised to achieve the objectives of this project. It also presents the methodologies utilized during the development phase and introduces a risk mitigation plan.

Chapter **??** presents the steps taken in the planning phase of the MPCS project. It details the system's architecture, the entities diagram, the designed mockups of the games's interfaces, the automatic user interface generation model using templates, and the proposed model of generating user interfaces using a JSON configuration file.

Chapter 6 shows the infrastructure and communication channels between the between each MPCS dissertations' project in the game, the progress acquainted in each sprint, the details and results of the usage of the automatic user interface generation using templates, and the details and results of the proposed model of generating interfaces using a JSON configuration file.

Chapter 7 presents both evaluations, the one to the MPCS project and the one to this dissertation's project, which was done with user studies, and the discussion about the results obtained from those.

Chapter 8 presents the conclusions about the objectives of the dissertation's project, the work done, the evaluations and results obtained, and future work.

Chapter 2

Background

describes how the various public/national authority' entities involved in marine pollution control perform and train to combat pollution events and will report the detailed usage of the simulator.

The entities in charge of marine pollution control are responsible for intervening, controlling, and combating pollution events. Combating Maritime pollution requires means consisting of the following [1]: (1) Equipment, such as booms, skimmers, pumps, air compressors, vehicles, ships, boats, and aircraft, suitable for the most common types of HydroCarbons (HC) spills, (2) People, professionals trained and competent to operate this equipment, continually adopting adequate procedures, whether in terms of personal, environmental and material safety or terms of efficiency and economic cost, means that are geographically, administratively and functionally distributed by (3) Facilities, belonging to (4) Organizations, public and private entities referenced in the Clean Sea Plan.

To be prepared to combat pollution, there is a need to be in a high state of readiness. To accomplish this, regular practical training is required. This training consists of the following [1]: (a) Coordination of the means involved (direction, actions, sequence and timeliness of actions, information management, means management, security, and effectiveness), (b) Communication (information, orders, clarity), (c) Procedures (individual and teams) such as Logistics (movement of means), Technicians (equipment operation) and Legal, national and international administrative (appointments, information, request for support). There is also the need to keep the equipment maintained. The Central Services of Central Services of the Directorate General of Maritime Authority (DGAM) and the Directorate for Combating Pollution of the Sea (DCPM), in particular, are aware of all the sea pollution events from all possible sources and are responsible for informing the appropriate entities [1].

The Maritime Authority System (MAS), upon learning of the spill, should provide the appropriate response, taking into account the following criteria [1]: (1) Safeguarding human lives at risk; (2) Do not unnecessarily put human lives at risk; in particular, it must be stressed that personnel can only start operations if they are properly equipped; (3) Reduce damage to property (goods) and the environment; (4) Do not cause unnecessary damage to property (goods) and the

environment.

2.1 Execution

The plan of execution to combat the oil spills is divided into three parts, preparation, combat, and investigation.

In preparation, the actions aim to confirm the event, inform the competent entities and define the command and intervention structure. These actions occur, preferably, before the combat and are the following [1]:

1. **Definition of the entity responsible for directing and coordinating maritime pollution response**: where the designation of the pollution response coordinator depends on the nature, location, and severity of the spill event, and is chosen by the Director General of the National Maritime Authority (NMA).
2. **Establishing the Degree of Readiness**: when the pollution event is identified. The responsible entities establish the degree of preparedness appropriate to the situation.
3. **Information from the Responsible Entities**: It is important to make the pollution episode known to other entities interested in the matter, according to the urgency and their attributions and competencies.
4. **Activation of the Command and Control Center (CCC)**: This presupposes the appointment of technical advisors, the organization of communications, and the distribution of responsibilities.
5. **Activation of the 1st Intervention Team(s)**: This is the team that performs the first response to the pollution event.
6. **Start Recording the facts and actions**: This is the first action to take. It is a document to record time facts and actions, including the record of all material and human resources.
7. **Collecting Samples and Sending them for analysis** to determine the pollutant and its subsequent condemnation.
8. **Public Relations Officer Appointment** that is responsible for contact with the Media.
9. **Liaison Officer Appointment**: In the case of a significant pollution incident or an incident, which may also affect neighboring States, the competent maritime authority must designate a liaison officer to expedite contacts with other bodies involved, including those of neighboring States, as appropriate, and inform them accordingly.

The combat actions aim to reduce the risk of a spill or containment or eliminate its environmental and economic effects. These actions consist of the following [1]:

1. **Situation Assessment**: There the authority directing and coordinating the operations must always take into account the main points when assessing the situation to define or correct the course of action to pursue, such as dispersing or collecting the spilled HC the fastest way to minimize the damage it can cause to marine life and economic activities and prevent them to reaching coastal spaces because, if it gets there, it is a very costly and labor-intensive operation.

2. **First Intervention Team**: must be mobilized to the pollution event site, to try to minimize the damage and begin pollution response operations.

3. **Intervention by DCPM**: It can support maritime authorities if requested by them and authorized by the Director General of the Maritime Authority. They can provide technical advice, support with equipment, or even reinforcement of the teams in terms of personnel and material.

4. **Safety and health of the intervention teams**: This is the first priority, and to accomplish this, the teams should use the appropriate protective equipment, execute the procedures safely and have the guarantee of adequate food, rest and medical support.

5. **Civilian health and safety**: This is the priority of combat actions, saving human lives and isolating areas at risk for their safety.

6. **Material safety and health**: The material must be in perfect condition to use; all general safety rules must be followed, as well as safety rules and operating procedures advised by the manufacturer.

7. **Containment and Guidance of the spill/Protection of sensitive areas** (harbor entrances, water intakes, aquaculture, nature reserve). This is the first objective of pollution response to prevent further damage and facilitate pollutant collection.

8. **Collection, Cleaning, and/or Dispersion**. In the sea are done by skimmers. Some types of reclaimers can be used on land, but the collection is mainly by human labor.

9. **Temporary Storage**: On the sea, the collected materials may be stored temporarily in a safe and suitable place, like spill recovery vessels or floating tanks, until appropriate and safe recycling or disposal can be planned, like a ship. On land, as the polluted site can be difficult to access for motor vehicles, the collected pollutants are stored in rigid bags or containers waiting to be collected or carried by hand to the vehicles that will transport them to the places that the municipal and environmental protection authorities indicate for temporary, adequate and safe storage.

10. **Final destination**. Pollutants collected at sea and on the shore are temporarily stored but must then be transported to treatment sites or durable storage.

The final part, the investigation, aims to determine the causes and responsibilities of the vent. These actions consist of the following [1]:

1. Record of facts and actions.
2. Identification of ships/vessels and persons involved.
3. Visit the event site to collect evidence.
4. Witness Interviews.
5. Forensic Analysis.
6. Analysis of the spilled HC.
7. Cost evaluation.

2.2 Simulator

To lower the high costs of real-time training, train more frequently and better prepare the entities to combat the pollution events, the opportunity to construct a simulator to perform real-time scenarios for training and evaluating the trainees to respond to future pollution events appeared. The creation of this simulator is a good concept since, as it was already noted, it saves much money and allows users to train more frequently because it is less expensive, which helps them be more prepared to act.

The simulator's main focus will be training, giving users a virtual experience whose realism is both required and adequate for efficient learning and training in its transfer to a real-world environment. After logging in, the user can interact with other users and carry out the same tasks as they would in the real world.

The mathematical model operating in the Simulator will mimic the Spill based on computed hydrodynamic and climatic data seen in the recent past, the site where the exercise occurs, and the military activities carried out. Every user will behave and act in the same manner as they would in their respective real-world roles.

There is a configuration phase where the user can configure the event and its evolution in time, which depends on the sea's state, the atmosphere, the tides, and the combat actions. These configurations constitute relevant information for the Marine Pollution Combat. The simulator has two modes: exercise and teaching.

In exercise mode for training, the simulation will not have any aid or tips to help the user. At the end of the simulation, there will be an individual and team performance evaluation related to the combat, stating the actions that were done well and the actions that were mistakes so that the user can learn and do better in future simulations. All the user actions will be saved in history so that a manual review can be done for evaluation purposes. The team performance evaluation will be based on an automatic analysis based on performance indicators. Also, the Event setup parameters should be created mostly randomly.

The performance evaluation will be made based on the following criteria [1]:

Objectives	**Stop criterion (end of Fiscal Year)**
Collect as much spilled HC as possible in x hours of Exercise	The first that occurs between x hours of Exercise or all spilled HC being collected
Contain the largest possible volume of HC spilled in x hours of Exercise	The first that occurs between the expiration of x hours of Exercise or when all spilled HC has been contained
Perform the Preparation phase in the shortest possible time up to x hours of Exercise	The first that occurs between the end time of Preparation and x hours of Exercise

Table 2.1: Performance evaluation

The teaching mode, as the name says, consists of simulating with the purpose of the user learning how to use the simulator. Aids or tips are available to help the user to learn how to act and interact in the simulation. Three types of e-learning courses are to be implemented in the Marine Polution Control Simulator (MPCS) project [1]:

1. E-learning courses to teach regulations, legislation, technologies, methodologies, and leadership in Marine Pollution Control (MPC Knowledge).
2. E-learning course to teach how to operate the Simulator (Simulator Operation).
3. E-learning course to teach how to manage the Simulator (MPCS Management).

Figure 2.1 shows an example of a possible gameplay. We can see that the user has access to his main stats and several lists of information he may need to work with. We can also see the interaction between the user and the equipment to access the main stats of the equipment.

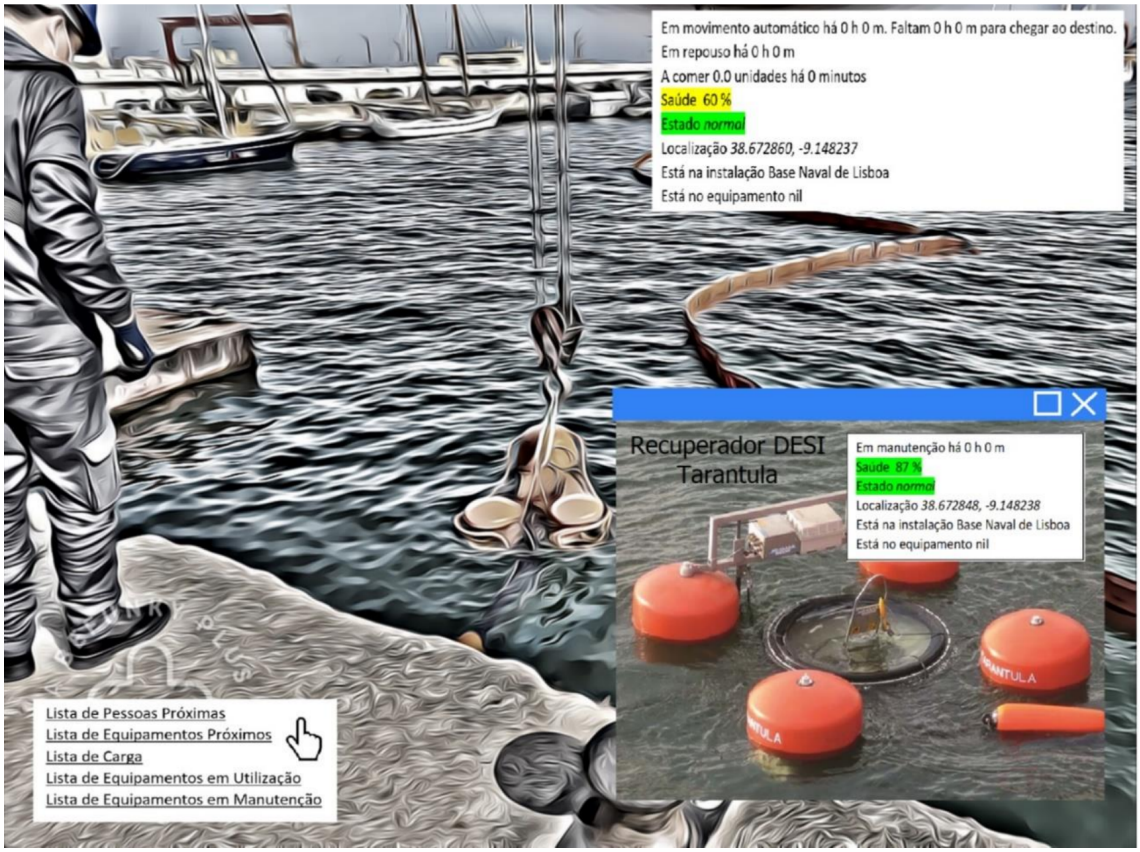


Figure 2.1: Example of a possible gameplay [1]

Table 2.2 lists the general and fundamental functional requirements of the simulator [1].

ID	Description
1	The Simulator should allow time compression, managed by the MPCS Manager, which should be automatically stopped as soon as any user performs any action or when any automatically running action ends. The MPCS Manager should not have access to time compression while actions are being performed by different users.
2	The Exercise should be able to be stopped and resumed by the MPCS Manager.
3	Whenever there is a time compression, or interruption of the Exercise, all users should be made aware of the situation and the reasons.
4	The MPCS Manager, during an Exercise, should be able to communicate with each user, or all users, through a pop-up window that should open on each user's computer.
5	Throughout the Exercise, there should always be the possibility for the user to press the F1 key for help in operating the Simulator and the F2 key for help/knowledge on Marine Pollution Control.
6	The Maritime Pollution Combat aids should exist in the base version, but they can be changed and others can be created by the MPCS Manager on his own initiative and/or at the request of the different entities that combat maritime pollution and that naturally participate in the exercises. These aids should be able to be in text, image and/or 2D and 3D video/animation.

ID	Description
7	The mathematical model should provide the simulation, for each instant of time, with all relevant spill data (type of HC, physical and chemical properties of HC, leak rate, volume, spill area, ...) and the hydrodynamic and meteorological database should provide the simulation with the state of the sea (current speed and direction, water temperature, swell, ...) and the state of the atmosphere (wind strength and direction, air temperature, precipitation, ...). The hydrodynamic and meteorological data should be actual data that actually occurred in the time interval, some years ago, and in the geographical area in which the Exercise takes place.
8	In marine pollution response, it is important for the CCC to forecast the drift of the spill, the sea state and the atmosphere. The drift forecast is, of course, made on the basis of the available information of the spill and the state of the sea and the atmosphere at the location, as well as the forecast of the evolution of the meteorological conditions. There are entities, such as IPMA (Portuguese Institute of Sea and Atmosphere) and IH (Hydrographic Institute) , that can provide weather forecasts. In the simulation, the forecasts provided by these entities to the CCC will be simulated based on the actual data from the hydrodynamic and meteorological database, but with a random error, variable depending on the time interval considered, to give realism to the forecasts.
9	For all user actions, or commands, on the person, other people, facilities, and equipment, the Simulator should display a 3D animation illustrating the action that the user initiates and another that ends. (For example: if the action is to get into a car, the animation should show a person getting into a car).
10	In the Simulator, a naval or aerial means is a piece of equipment. In each installation to which that means belongs, there should exist, at least, a virtual person that will operate (use) it, as, for example, setting course and speed, putting in the sea a semi-rig or communicating with the CCC, collecting samples from the HC, etc....
11	A naval or aerial means, when approaching the Event and/or Spill site, should automatically report (without user intervention), depending on its location and spill evolution, the information provided by the mathematical model of the MPCS. The report can only contain information that in a real situation would be accessible, that is, if the naval means is at a distance that in reality would allow it to visually observe the spill and the accident and to contact the ship's commander, it can inform the approximate area of the spill (this area will be calculated by the Simulator multiplying the area, provided by the mathematical model, by a random error), if there are crew members in danger, if the accidented ship has maneuvering capacity, etc... Likewise, the response to a request for analysis of the spilled HC should include the characteristics of the HC, defined in the Event, as if they were the result of the analysis by an analytical laboratory.
12	In the automatic movement of people and equipment by land, the Simulator should follow the fastest route suggested by a map's platform.
13	The hydrodynamic data and meteorology of the Exercise should be based on actual data but which occurred in the recent past.
14	The effect of combat actions on the spill should be calculated by the mathematical model adopted and/or developed.

ID	Description
15	People should have their health depend on their working time, their rest time, their food, and accidents. People may be injured or killed. Each person should have a cost of Hxh (Man x hour) in euros.
16	Equipment should have its health dependent on its working time, maintenance, and accidents. Equipment may break down or be lost (beyond repair). The performance of equipment should be dependent on its health, its proper use, and its technical specifications. Each type of equipment should have an Exh cost (Equipment x hour).
17	Only people should be able to move in manual or automatic mode. All equipment of the means of transport type should only be able to move in automatic.
18	All entities must be able to be georeferenced with a resolution of 0.3 m.

Table 2.2: Functional requirements [1]

Chapter 3

State of The Art

This chapter presents some concepts needed to understand the Automatic User Interface Generation (AUIG). It conducts a study about User Interface Description Language (UIDL)s and approaches that use Automatic User Interface Generation Methods (AUIGM). It shows a comparison between the studied approaches and a review of the learned things that may be used to solve our problem.

3.1 Concepts

This section presents some concepts to be considered to give context to the follow-up of the document.

3.1.1 User Interface

The means of human-computer interaction is called User Interface (UI). The definition of UI is the point of interaction and communication between a Human and a computer. The UI should be simple and intuitive so that the user can use the application best.

3.1.2 User Interface Models

A UI model is a framework for understanding and designing how users interact with a particular system or device. It is a crucial element of user experience design, as it determines how users will navigate through and utilize the features of a product or service. They attempt to formally describe the tasks, data, and users an application will have and then use formal models to guide the generation of the UI [2]. Several different UI models [2] have been developed over the years, each with unique characteristics and capabilities. Some examples of UI models are the following:

1. **Application model**: Defines the capabilities of the application, such as de-

scribing various classes in terms of their attributes, exceptions that methods may throw, methods together with their preconditions, and a list of events published by the class [3].

2. **Task model**: It describes the task to be accomplished by the user [4]. This model includes such tasks as, for example, preventing information to the user, obtaining information from the user, and invoking application functionality [3].

3. **Domain model**: The domain model describes the objects the user manipulates, accesses, or visualizes through the UIs [4].

4. **Presentation model**: The presentation model contains the static representation of the UI [4] by describing how the information will be presented to the user regarding low-level elements such as buttons, and menus. [3].

5. **Dialogue model**: Holds the conversational aspect of the UI [4] by describing the tasks that the user can perform with the system [3].

3.1.3 User Interface Description Languages

A UI description language is a high-level computer language used to describe the structure and behavior of a graphical user interface UI concerning the rest of an interactive application [4]. Determining a syntax (how these features might be articulated in terms of the language) and semantics for such a language is necessary (what do these characteristics mean in the real world). It can be viewed as a typical approach to UI specification, independent of the programming or markup language used to implement the UI [4]. The objective is to provide a way to describe the UI in a way that is independent of the underlying hardware and software platforms so that the same UI can be rendered consistently across different devices and operating systems [4]. Some common examples are HTML, XML, and JSON. Section 3.2 presents a more in-depth description of these languages, and a comparison is performed between some examples of these languages.

3.1.4 Automatic User Interface Generation

AUIG refers to generating UIs based on the context of use. It separates the UI from the application logic [5]. These methods are context-sensitive [5]. Contrary to manually creating the interfaces, they do not previously describe the interface to be generated; instead, they receive a description of the context and generate the interface in real time based on that description. The interface could use familiar elements the user has seen recently [5].

3.2 User Interface Description Languages

A UIDL specifies the interface to be generated.

The primary issue that prompted the need for UIDLs was the requirement to create a UI that functioned as a module of an interactive program modeled by a set of specifications in order to share and communicate these specifications with stakeholders, or to (semi) automatically generate the code of the UI, as desired in model-based approaches for developing UIs [4]. Also, this need grew with the portability issues; that is, when a UI was required to run simultaneously on different computing platforms, this need took form in some language that would be exchanged from one platform to another without any changes to avoid any extraneous development effort [4].

Many approaches have emerged to solve the portability problem of UI on different platforms. Some of these approaches are listed below:

- **Binary emulation**: Allows an application to be used on different platforms without recompiling, thanks to a software emulator.
- **Virtual toolkits**: Have been introduced to reduce the development effort: the developer writes a unique code using an API executed on all platforms where the API exists. It does not accommodate many platform constraints. This can be performed by actualization, where the toolkit binds the virtual created object to the real platform by actualizing them. The main benefit of this approach lies in the large range of virtual primitives. Nevertheless, its usage is limited by a massive run-time library. It can also be done by re-implementation, where the toolkit re-implements each virtual object for each platform. Although these tools contain some abstractions, they do not accommodate many platform constraints.
- **Ported APIs**: Support native APIs (usually Windows) on other environments.
- **Tools generating adaptive UIs**: Tools that generate a UI that can be adapted at runtime depending on the context of use.
- **Multi-context tools at the logical event**: Generate at design time a concrete UI for a specific context, from an abstract description of the UI. The abstract description is written in a specific language that differs from one tool to another.

Multi-context tools at the logical event is the approach related to this thesis. UIDLs describe one or more aspects of the UI: the static structure of the UIs, that is, the description of the UI elements and their composition; the dynamic behavior, including event actions and behavioral constraints; and finally the presentation attributes [6]. Following it is presented some examples of UIDLs and their respective description:

- **UIML**: It's a meta-language that allows designers to describe the UI in generic terms and to use a style description language to map the UI to various operating systems, languages, and devices [4]. UIML document contains a

UI description, a peers section that defines mappings from the UIML document to external entities, and a section used as a template that permits the reuse of previously authored material. The interface description is then produced following the presentation component's specifications and uses logic definitions to interact with the application logic. The renderer interprets UIML or compiles it into another language on the client device. The limitations of this language include the fact that it only provides a single language for defining various user interface types and that it does not permit the development of UIs for various languages or devices from a single description, necessitating the design of separate UI for various devices [4].

- **AUIML**: It's a language that focuses on defining the intent of an interaction with a user instead of focusing on the appearance. Designers have to concentrate only on the semantics of the interactions. It is intended to be independent of any client platform, implementation language, and any UI implementation technology. A single intent should execute on numerous devices and is designed to be independent of any client platform, implementation language, and UI implementation technology [4]. UI is described in terms of manipulated elements (a data model that organizes the information needed to support a specific interaction), interaction elements (a presentation model that specifies how the UI will look, such as choice, group, table, or tree), and actions that allow describing a micro-dialogue to manage events between the interface and the data. The presentation model offers freedom in the level of detail anticipated by the renderer in addition to the specification of the UI's appearance: the designer may choose to can either decide to precisely control what is to be displayed or only specify the interaction style, leaving the decision to the renderer [4].

- **XAML**: It's a language developed by Microsoft, used in many of their technologies, that defines the structure and layout of a UI. It is available not only on Windows computers but also on windows-based mobile devices. Developers need to define the layout elements and provide the policies in order to adapt (to a certain extent) automatically the Graphical User Interface (GUI) to the specific device [7].

- **XIML**: It's a language that provinces a way to describe a UI without worrying about how to implement it. The task component, which captures the business process and user tasks the interface supports, is one of the five fundamental interface components that XIML predefines. The user component captures the characteristics of the (group of) users who can use the application, the domain component is a collection of all the objects and classes used, the dialog component determines how the user interface interacts with the user, and the presentation component are the other components [4]. A XIML description comprises characteristics and relations in addition to the interface components. An attribute is a feature or property with a value and is a part of a component. There is already a set of predefined properties. A relationship connects one or more components(s) within the same model component or across several ones.

- **HTML5**: Its a markup language for structuring and presenting content on

the Web [7]. While it can be considered content-oriented, it offers several form tags to interact with the user, making it useful to define UIs [7]. This technology can be beneficial because of the usefulness of the client-server model to allow frequent content updates rather than providing application updates [7].

- **QML**: It's an UIDL that has been adopted by Ubuntu OS applications. It is a JSON-like language where the graphical elements are grouped in libraries that can be imported as needed [7]. It provides layout mechanisms to support device adaptation, which makes adopting this language a feasible solution for multi-platform development [7].

Now that some examples of UI description languages were introduced with a description, this paper presents a general comparison between them, presented in Table 3.1, using the following criteria:

- **Target**: It refers to the target the language was designed for, for example, mono/multi-platform, mono/multi-user, mono/multi-environment.
- **Supported languages**: This criterion refers to the languages that can be used with UIDL to support the application logic.
- **Supported platforms**: This criterion refers to whether the UIDL can be executed in different platforms with different characteristics (for example, different screen sizes).
- **Layout support**: It refers to whether the UIDL allows developers or not to select predefined GUI layouts.

Table 3.1 evaluates the referred UIDLs, using the metrics of comparison defined.

	Target	Supported platforms	Supported Languages	Layout support	Pros	Cons
UIML	Multi-platform [4]	Multi-device	JAVA, C++, WML, PalmOS [4]	Yes	- Easy to use	- Obsolete
AUIML	Multi-platform (available interactors, displays) [4]	Handheld devices, such as PGA, and desktop PC	Java Swing, PalmOS, WML [4]	Yes	- Independent of any platform, implementation language	- Developed for internal use in IBM, not publicly available [4]
XAML	Multi-platform	Multi-device, as long as it is a Windows-based device	Programming languages under the .Net platform (like Java, C++)	Yes	- Widely used	- Mostly used on Windows platforms
XIML	In theory, multi-platform, multi-user, and multi-environment; In practice, multi-platform	Multi-device	WML, Java [4]	Yes	- Predefines any concept model	- Obsolete
HTML5	Multi-platform	Any device, but requires a web browser	CSS, JavaScript	No	- Ubiquity of web browsers, the usefulness of client-server model to allow frequent content updates (rather than application updates) [7]	- Does not have layout support, so without using the stack HTML+CSS+Javascript, is kind of incomplete, raising the development effort to have full potential
QML	Multi-platform	Multi-device	JavaScript	Yes	- Provides layout mechanisms in order to support device adaptation [7]	- Does not offer a high level of customization

Table 3.1: Comparison evaluation of the referenced UIDLs

All the mentioned UIDLs are supported to be used in a multi-device environment; however, HTML requires a Web browser to be available in each device [7]. While suitable for developing GUIs for several platforms, XAML is restricted to work within Windows devices. Regarding programming languages, as for QML, Javascript is recommended to be used in Ubuntu OS, but many more languages can be used using Qt libraries. Also, for HTML, the application logic is developed in Javascript. For XAML, the support languages are available under the .NET platform [7]. Regarding the layout support, all UIDL have it except for HTML, which needs help from CSS to layout the structure of the interface [7].

Overall, AUIGM can use UIDLs to specify the desired layout, appearance and the behavior of the generated user interface, potentially saving time and effort for the developer. The UI generator can then use this description to create the user interface automatically.

3.3 Automatic UI Generation Approaches

This section focuses on AUIG approaches and evaluate them using comparing methods.

3.3.1 The Personal Universal Controller Approach

The objective of this project was to use handheld devices the users carry, such as PDAs and mobile phones, to remotely control homes' and office appliances, such as televisions, microwave ovens, and copy machines [5].

To accomplish this, they would need to develop multiple interfaces, each for a kind of device. This is a major problem because it creates a lot of design effort because of the variety of devices possible to use with this system. To fight this problem, they studied automatic user interface methods and implemented them to solve this problem.

The authors state that using AUIG, the UI can be context-driven by incorporating the user's specific information and current situation into the design of the UI. For example, the UI can be displayed on the user's device, or the UI can display familiar elements that the user has seen recently.

Figure 3.1: A diagrammatic overview of PUC the system [5]

The Personal Universal Controller (PUC) project is applying model-based techniques to automatically generate remote control interfaces for all of the computerized appliances in the environment and is exploring how the generated interfaces can be automatically customized to both the control device and the user. When a user requests an interface to control a device, the user's appliance downloads a functional model and uses it to generate an interface automatically [5]. A functional and concise XML-based language for modeling the appliance's functions was designed, and rules for generating user interfaces from their language on several different devices.

These rules define how the generated UI will be based on the appliances' description. These descriptions include a few parameters such as data type information, for example, boolean, integer, enumerated; state variables, that is, a variable that represents the alter the state of the appliance, for example, for a DVD, the power button; commands that represent more complex components that cannot be represented by state; label information that represents the interface components that represent state variables and commands; group trees that layouts the interface by aggregating similar elements in the interface and keeps far way different elements, for example, the play and stop button being together; and finally dependency information that contains information about the dependencies of an interface, that is, a state variable is disabled depending on the values of other state variables.

With all this information in the description of an appliance, they created a set of rules that, depending on the information, creates different XML-based interface

descriptions that are then generated by a graphical interface generator.

The workers state that this approach was not successful for general application interfaces but that it can be a valuable tool in specific situations. They believe that AUIG is essential for supporting User Interfaces on multiple devices and incorporating situational information into interfaces. They encountered challenges such as finding specific domains where automatic generation was possible, improving modeling techniques and combining models to improve usability [5].

3.3.2 SUPPLE: Automatically Generating User Interfaces

The authors state that to give people ubiquitous access to software applications, device controllers, and Internet services, it will be necessary to adapt user interfaces to the computational devices at hand automatically [8]. A critical aspect of this vision is the premise that every application (whether an email client or room-lighting controller) should be able to render an interface on any device at the user's disposal. Furthermore, the rendering of an interface should reflect the needs and usage patterns of individual users. Given the wide range of device types, form factors, input methods, and personal needs and interaction styles, it is unscalable for developers to create interfaces for each type of device and every kind of user. Instead, an automated solution is necessary [8].

To solve this problem, they developed a system called SUPPLE, which treats interface generation as a combinatoric optimization problem. The combinatoric optimization is based on a functional description of a UI, which says what functionality the interface should expose to the user, and takes into account both device and user properties. When this system is asked to render an interface (specified functionally) on a specific device for a specific user, the system searches for the rendition that meets the device's constraints and minimizes the estimated cost (user effort) of the person's activity. The UI generation process is an optimization problem where the algorithm tries to minimize the estimated overall effort [8]. Figure 3.2 illustrates the formal specification of the classroom interface.

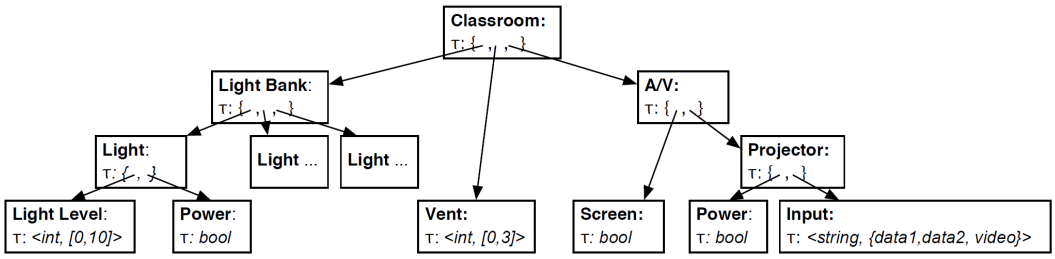


Figure 3.2: Tree depiction of the functional interface specification for a classroom appliance controller [8]

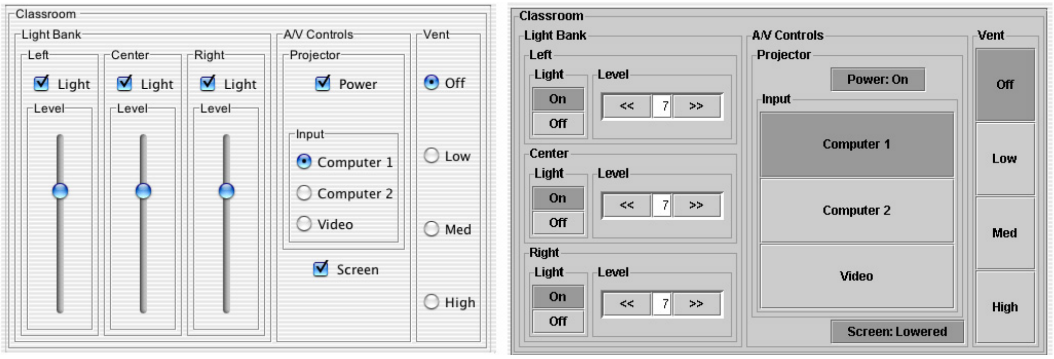


Figure 3.3: The classroom interface rendered for two devices with the same size: (on the left) a pointer-based device (on the right) a touch-panel device [8]

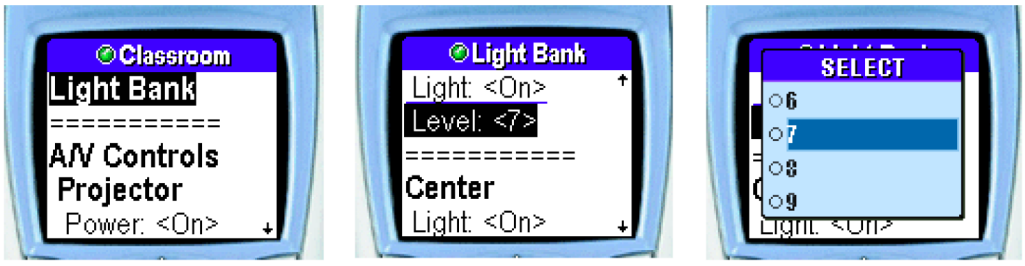


Figure 3.4: The classroom interface rendered on a WAP cell phone(Sony Ericson T86i) [8]

As we can see, the approach takes on the functional interface specification and renders a different user interface for each type of device. This approach reduces the design effort, as we can conclude by observing Figures 3.3 and 3.4, where one implementation, depending on a functional interface specification, worked to generate interfaces for three different devices. Doing this manually would increase the design effort significantly as they would have to design each interface isolatedly.

3.3.3 ICrafter: A Service Framework for Ubiquitous Computing Environments

ICrafter is a framework for services and their user interfaces in a class of ubiquitous computing environments [9]. The authors describe an interactive workspace as a technology-rich space with interconnected computers, like conference/meeting rooms and classrooms. The objective of this system is to allow users of an interactive workspace to interact with the services presented in this space, such as projectors, lights, scanners, or applications like PowerPoint. The users interact with these devices using various access/input devices, like laptops, and handheld devices. These input devices are called appliances.

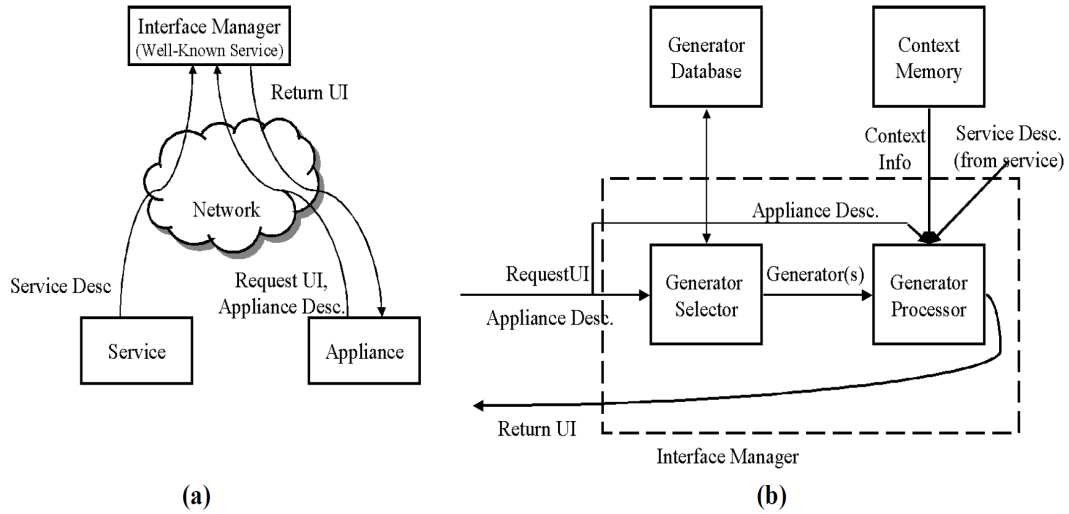


Figure 3.5: ICrafter architecture [9]

Appliances provide a self-description and ask the Interface Manager for User interfaces. Based on the requesting appliance and the services for which the UI was requested, the Interface Manager initially chooses the suitable UI generators. In order to generate the UI, it then executes the generators with access to the service descriptions, appliance descriptions, and context [9].

ICrafter uses unique UI generators (target platforms) for various services and UI description languages. A template system is used to implement the majority of them.

The use of so-called service patterns is an intriguing concept introduced by the ICrafter approach. The services offered exhibit patterns that can be identified. Then, ICrafter produces UIs for these designs. On the one hand, this facilitates easier service aggregation and higher consistency. On the other hand, the aggregated service does not offer distinctive features. ICrafter, to create a UI for a projector control works the following way: (1) Have a context memory with all the devices and services that need the information of; (2) For each service, has UIDL representing the service; (3) Then have the services' templates written in HTML; (4) For the final step, they combine the information from the service UIDL and the respective service template, and with both information generate the corresponding HTML, that is then rendered in the browser.

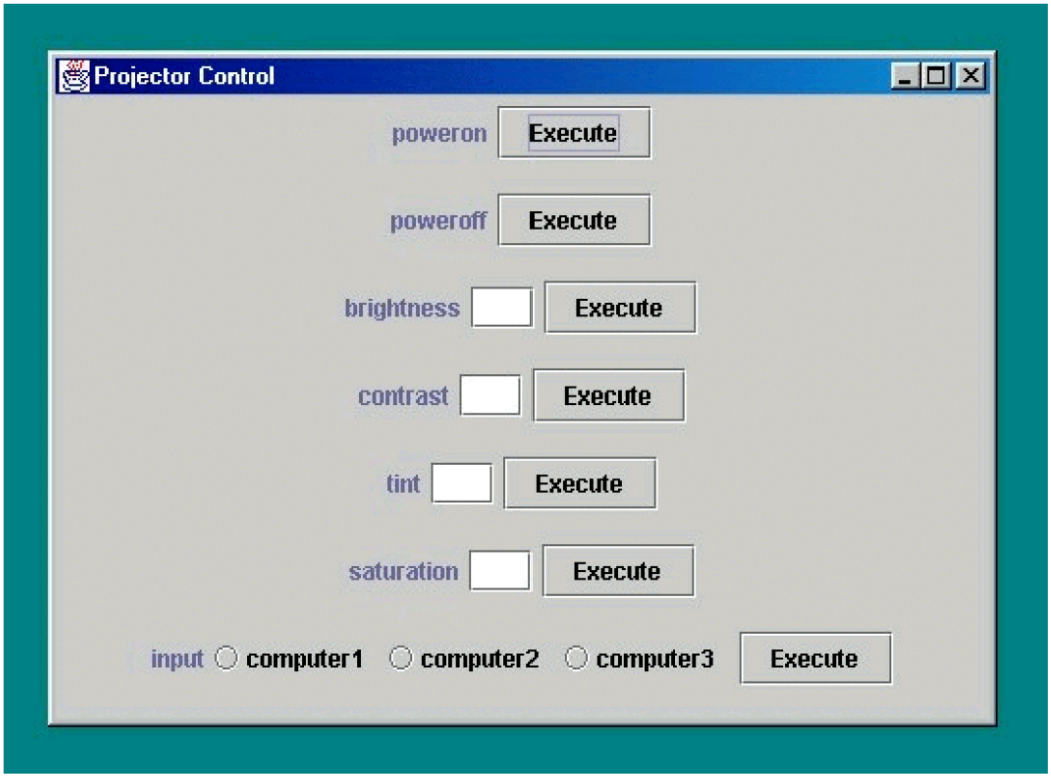


Figure 3.6: Interface designed by ICrafter to control the projectors of a room [9]

As we can see in Figure 3.6, there is an example of an interface generated by ICrafter for a computer.

3.3.4 Tool Support for Designing Nomadic Applications

They present a tool called TERESA that supports top-down transformations from task models to Abstract User Interface (AUI) and then to UIs for different types of interaction platforms, such as mobile phones or desktop computers [10].

They state common problems that developers have when developing software for multiple different platforms, such as extra developing effort, expensive maintenance costs, and an increase in the problems related to configuration management.

Their approach considers three aspects: developers can define the input and output needs of their applications, vendors can describe their devices' input and output capabilities, and users can specify their preferences [10]. The general idea of this approach is to have an abstract UI description and then an environment that suggests the design of the UI based on the specific device and possible context of use.

This model-based approach is composed of four main steps, which are the following [10]:

- **High-level task modeling of multi-platform application**: In this phase,

the developers create a single model containing all possible contexts of use, roles involved in the application, and a domain model aiming to identify all the objects that have to be used in order to perform the tasks and relations between all the objects. These models are created using the *ConcurTaskTrees* notation.

- **Developing the system task model for the different platforms considered**: Here, the designers have to filter the task model created based on each target device and, if necessary, refine it depending on the device considered, obtaining the system task model for the specific target device.
- **From system task model to abstract user interface**: In this phase, an analysis is conducted on the task model relations, aiming to obtain an abstract notation of the user interface to be composed.
- **User interface generation**: In the last step, there is a consideration of specific properties of the target device, and then every interactor is mapped into interaction techniques supported by the particular device configuration considered, such as the operating system or toolkit.

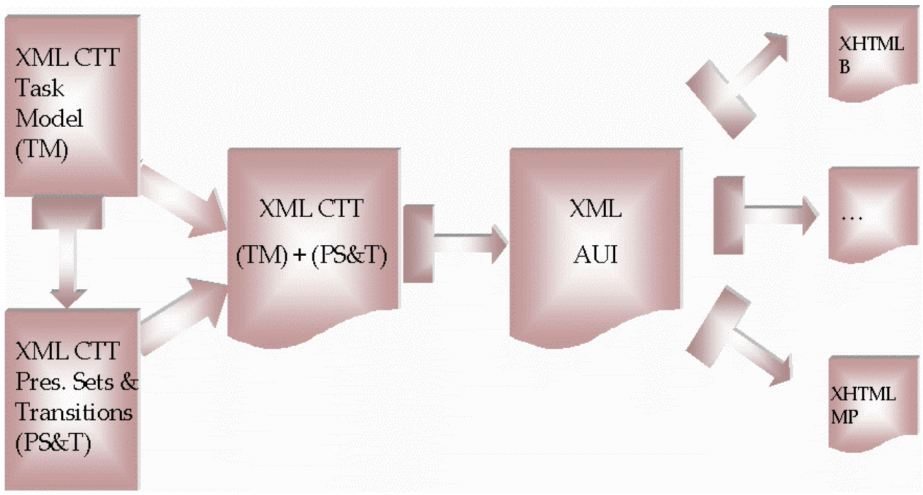


Figure 3.7: Main transformations in TERESA in terms of XML-based applications supported [10]

Analysing Figure 3.7, we can relate it to the four phases mentioned above. It is possible to obtain the set of tasks from the XML specification of a Concurrent Task Tree (CTT) task model. The XML task model and presentation set specifications are the input for the transformation generating the associated AUI. The created AUI yields the respective ConcreteUser Interface (CUI) according to the selected interaction platform. In the last step, the tool automatically generates the final UI, based on the concrete UI previously created.

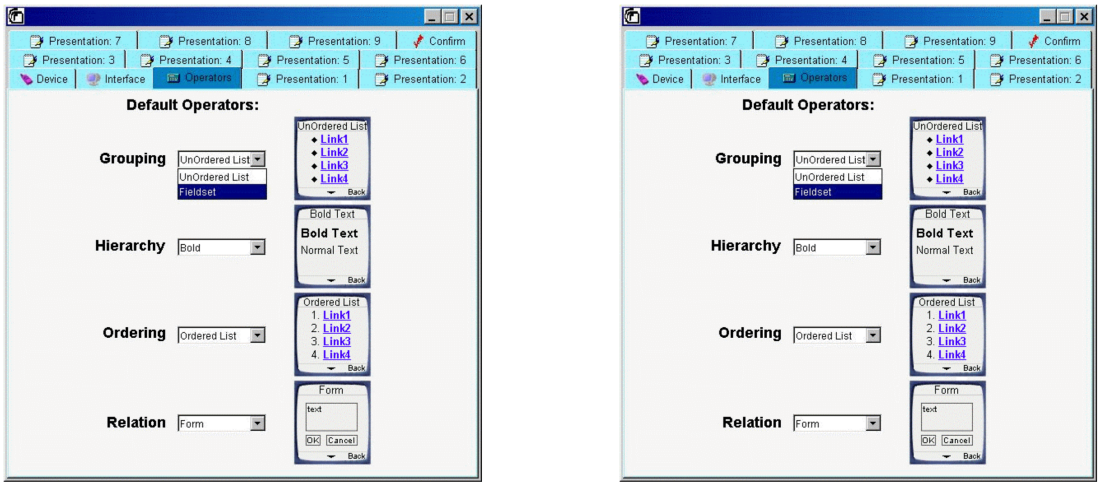


Figure 3.8: Configuring the presentation specifications in TERESA [10]

In Figure 3.8, we can see how the tasks presented in the task model will be converted into the concrete UI. Also, we can see how we customize how the concrete UI elements are going to be. Next, it presented the filter application on the task model for each type of selected device and the resulting user interfaces.

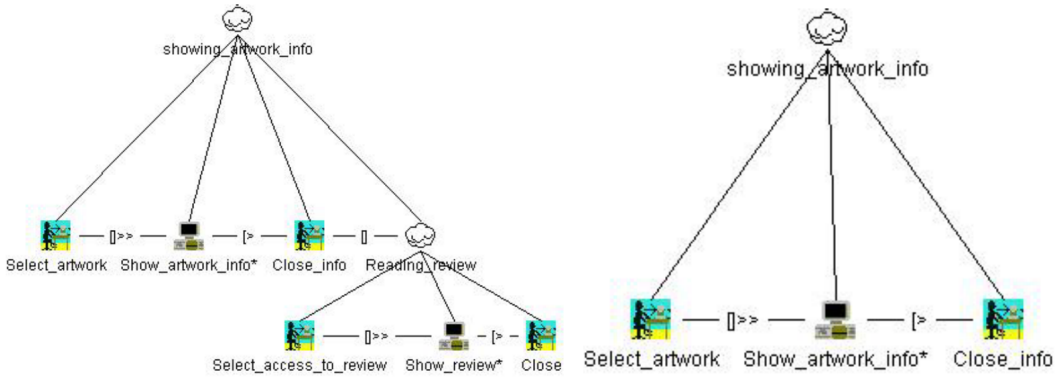


Figure 3.9: Task model after applying the filter for the desktop environment, on the left and for the mobile phone on the right [10]

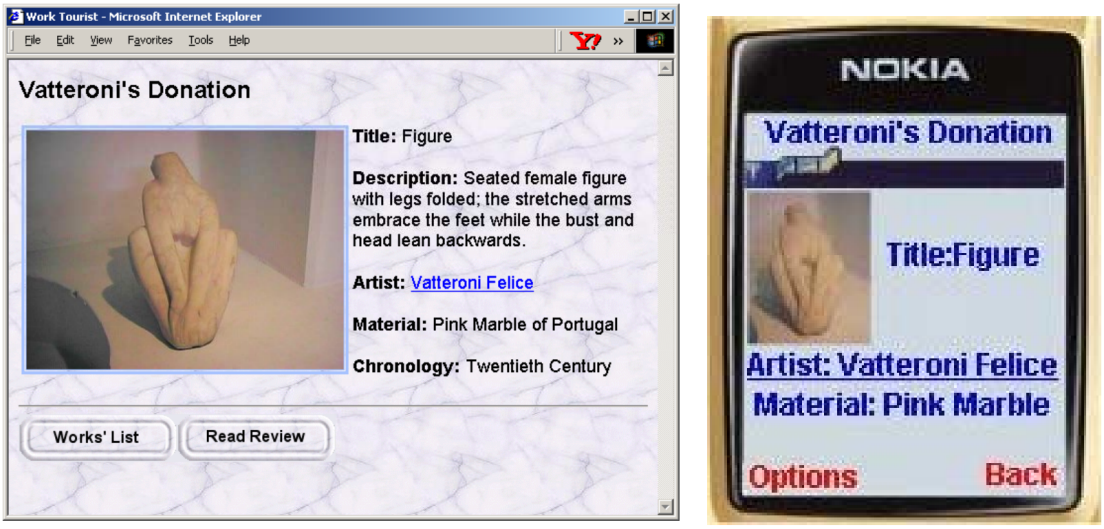


Figure 3.10: Resulting UI of the example from the task model [10]

In Figures 3.9 and 3.10, we can see how a filter applied on the application task model, for each type of device, desktop and mobile phone, makes the difference in the task model, and consequently in the resulting UIs, as expected. With this, we can conclude that the workers successfully developed a model-based approach, based on a task model, to create concrete UI from different platforms, in this case, a desktop computer and a mobile phone.

3.3.5 User Interfaces for Smart Things – A Generative Approach With Semantic Interaction Descriptions

In this approach, the workers believe that bringing smart devices into peoples' homes and enabling the user to have better interaction and be more intuitive with these devices, requires deployable interaction mechanisms [11]. The problem with this interaction is the need to create interfaces for every kind of smart device. To solve this, the workers wanted to generate UI based on the semantics of the interaction rather than providing an explicit concrete encoding of an appropriate type of UI or its appearance, reducing greatly the amount of time and work needed to create appropriate and easy usable UI for smart devices.

In order to enable the automatic generation of user interfaces, they developed a model-based approach where the smart device provides a description of how other devices can interact with it, in a form of a UIDL.

This proposed description technology makes use of data type information, but only to describe the exchange of data between the devices and not to specify the user interface component. This description consists of two different parts, one is the information about the high-level semantics of the interaction and the other one is the data type information for the data exchanged [11]. The high-level semantics information is organized in hierarchical taxonomies representing interaction abstractions. The data type represents the type of entity state for sensors

and stateful actuators.

Both this information together with a set of rules to process it, define a concrete semantic interface description language by embedding information for the appropriate data type and interaction abstraction as metadata into the interactive components of devices, as we can see in the example shown in Figure 3.11

```
{
  "type" : {
      "name" : "enum",
      "values" : ["down","stop","up"]
  },
  "abstraction" : {
      "name" : "move",
      "orientation": "vertical"
  }
}
```

Figure 3.11: Example of the generated interface description language to control a window blind in JSON [11]

This generated description can be expressed and embedded in multiple formats, such as XML documents, or as HTML-based Microformats [11]. Using these formats facilitates the change of information between the devices by using the web, preventing the information from being meaningful for the users and for the devices to be in different documents. Figure 3.12 represents the window's blind interface description in HTML microdata.

```
To <span itemprop="name">move</span> the blinds <span itemprop="
    orientation">vertical</span>ly, set the controller to one of
    the values <span itemprop="type-range">[down,stop,up]</span>
    (data type: <span itemprop="type-name">enum</span>).
```

Figure 3.12: Example of the generated interface description language to control a window blind in HTML microdata [11]

The workers implemented a prototype application for mobile devices running on Android system that interprets the interaction descriptions and allow the users to interact with devices via automatically generated interfaces [11], as we can observe in Figure 3.13

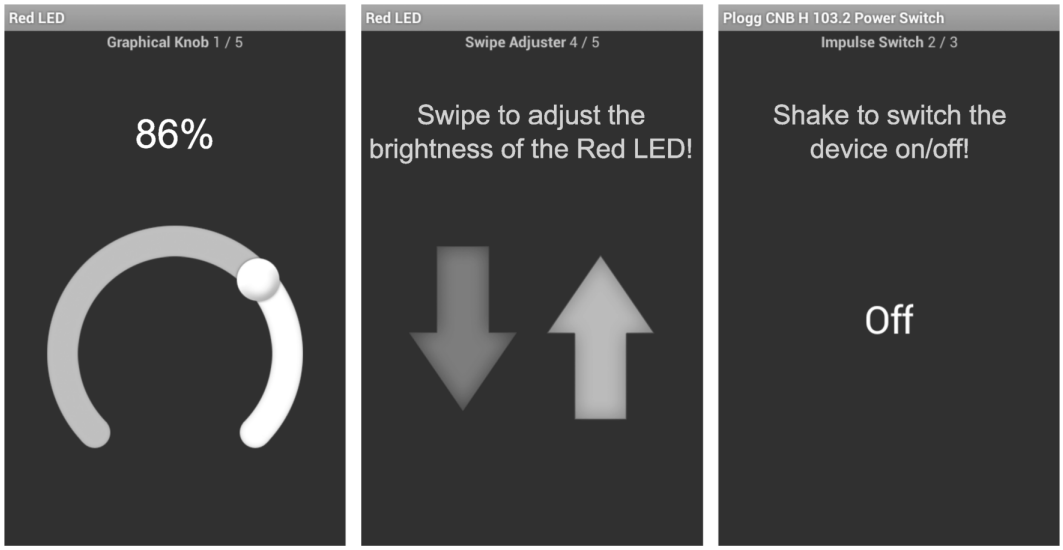


Figure 3.13: Smartphone interfaces for controlling the brightness of a LED on left and middle, and a power switch on the right [11]

3.4 Evaluation

In this section, we define some evaluation criteria and use them to compare the previously studied approaches in section 3.3 and perform an analysis.

The metrics for comparing evaluation that it is going to be used are the following:

- **Supported platforms**: This criterion refers to whether the UIDL can be executed in different platforms with different characteristics(for example, different screen sizes).
- **Principle**: This criterion refers to a small summary of how the solution was built in each approach.
- **Context-Sensitive**: This criterion refers to if this approach is context-sensitive or not.

	Principle	Supported platforms	Context-sensitive	Pros	Cons
PUC [5]	Rule-based transformation	Multidevice	No	- Multimodal - Multidevice	- Obsolete
ICrafter [9]	Template-based transformation, service aggregation	Multidevice	Yes,	- Multidevice - Well designed model-based solution	- Obsolete - Platform specific templates must be developed for each service/appliance - Limited to interactive workspaces
SUPPLE [8]	Combinatoric optimization	Multidevice	Yes	- Multidevice - UI generation combinatoric optimization	- Hard to support of multiple platforms
TERESA [10]	Task model that gets filtered for each device, and converts to an uidl that can be rendered	Cellphone, PDA, Desktop	No	- Easy customization of the rendered UIs - Automatic filtering for each type of device	- Low variety of possible devices. - Obsolete - designing effort by having to design the full task model of the application
UI for Smart Things [11]	Rule-based transformation, but context-sensitive	Multidevice	Yes	- Very adaptive to new unknown devices - Modular solution by aggregating possible UI based on respective data type	- Possibility of the rendered UI not being the most suitable

Table 3.2: Analysis on the studied approaches

Reviewing the previous comparison, we can get much important information that we may use in the development of the project. As mentioned earlier, we need to automatically generate interfaces for various interfaces. Although SUPPLE [8] and TERESA [10] are interesting approaches that use a functional model to generate the interfaces, we do not find them valuable enough for the development of this project.

The other three approaches have their strengths and weaknesses, for example, UI for smart things [11]; although it supplies the UI generator with a series of possible UI elements to generate, it may not have one that can be suitable enough for the user experience. On the other hand, this approach and PUC [5] use an interesting principle based on rules that, depending on the description, have a set of rules that modulate the UI based on the needs. The ICrafter also has an interesting approach that uses pre-created templates for each service and, together with the service description, generates the needed UI. ICrafter [9], instead of using a set of rules, uses a template-based transformation, where a set of templates are created, and according to the information provided in the device's description, the software chooses the most appropriate template for the provided description, and the UI is generated. This approach has an interesting solution that could be of great use in the future when all the possible UIs that will be required in the simulator are studied, but for now, by not knowing them, we can't use this technique, because we do not know how the templates will be. Although this is a more straightforward and more organized solution than PUC [5] and UI for smart things [11] (both approaches can get a very complex set of rules) having templates can also increase a lot the design effort when the number of different interfaces to be generated increases because it would be needed to create a template for a single/group of interfaces.

Chapter 4

Methodology and Work Plan

This chapter focuses on the methodologies and work plan used to develop this project. It defines and explains the techniques used in the development phase and the tasks necessary to accomplish the project's objectives. Also, it presents a risk plan, which shows the possible risks of this project and their respective mitigation plan.

4.1 Methodology

For the development of this project, the methodology used was Scrum. Scrum is a lightweight framework that helps people, teams, and organizations generate value through adaptive solutions for complex problems [12]. As Linda Rising and Norman S. Janoff say, Scrum is a development process for small teams [13]. This methodology is composed of the development team, which will work to develop the product, a Scrum Master, that is in charge of advising and controlling the developers and supports the Product Owner, that involves all stakeholders relative to the project and orders the work for a complex problem into a product backlog[12], that is a list of the features that the client wants to see implemented in the product and maximizes the value of the resulting product. Scrum has three pillars [12]:

- **Transparency**: The emergent work process must be visible to those performing and receiving the work.
- **Inspection**: The Scrum artifacts and progress to the agreed goals must be inspected frequently and diligently to detect potentially undesirable variances of problems.
- **Adaption**: If any aspects of a process deviate outside acceptable limits or if the resulting product is unacceptable, the process being applied or the materials being produced must be adjusted.

This methodology has an initial planning phase, where the team must define an architecture and the Scrum master. After the initial planning phase follows a

series of short development phases called sprints, where each sprint lasts about one to four weeks, with a goal, and the team delivers the product incrementally until the final delivery [13]. During each one of the sprints, there are several short meetings, usually weekly, where each team member addresses three questions [13]:

- What have you completed since the last meeting?
- What obstacles got in your way?
- What do you plan to accomplish between now and the next meeting?

These questions are discussed with every team member in every meeting, making it fundamental because it helps every member track their colleague's progress and vice-versa. These meetings are short, lasting for 10-15 minutes, and only serve to address the questions mentioned before and not to brainstorm a solution [13]. At the end of the sprint, the team produces an increment that builds on the previous increments, and everyone involved meets for the Sprint review to report the information to the client. In this meeting, anything can be changed, such as more or less work or even reprioritization of work. After this meeting, there is a sprint retrospective where improvements are discussed to be implemented in the next sprint.

Scrum was chosen as the methodology for this project because we found it the most suitable because of its transparency and robust features of planning and organization. Since there are some dependencies between the team members, we were able to know the progress of our colleagues and vice-versa and better plan and organize our work based on it. With the sprints and the weekly meetings, we could keep tracking the time needed to accomplish the sprint goal, making our progress more planned and organized.

Given the complexity of the project, these meetings also served to get guidance and brainstorm ideas to achieve the best solution possible. There was a lot of discussion about new ideas to implement, features to remove, and guidance for our dissertation. Every interface presented in the game's final version emerged from the weekly discussion.

4.2 Work Plan

Taking into account the objectives defined in section 1.3, a set of tasks was defined to better plan and organize the work needed to accomplish them, presented in Table 4.1.

Semester	Task
First semester	1 - Get acquainted with the MPCS project and definition of the document structure
	2 - Study automatic UI generation, UI models, UI description languages
	3 - Write intermediate report
	4 - Propose a UI generation approach and develop a proof-of-concept
Second Semester	5 - Design the system architecture and infrastructure
	6 - Creation of the UI mockups
	7 - Implement the proposed UI generation strategy
	8 - Assess the proposed UIs with user studies
	9 - Write dissertation

Table 4.1: Work plan for each semester

Figure 4.1 is a Gantt chart showing the duration, in days, taken into the accomplishment of each one of the tasks defined in Table 4.1.

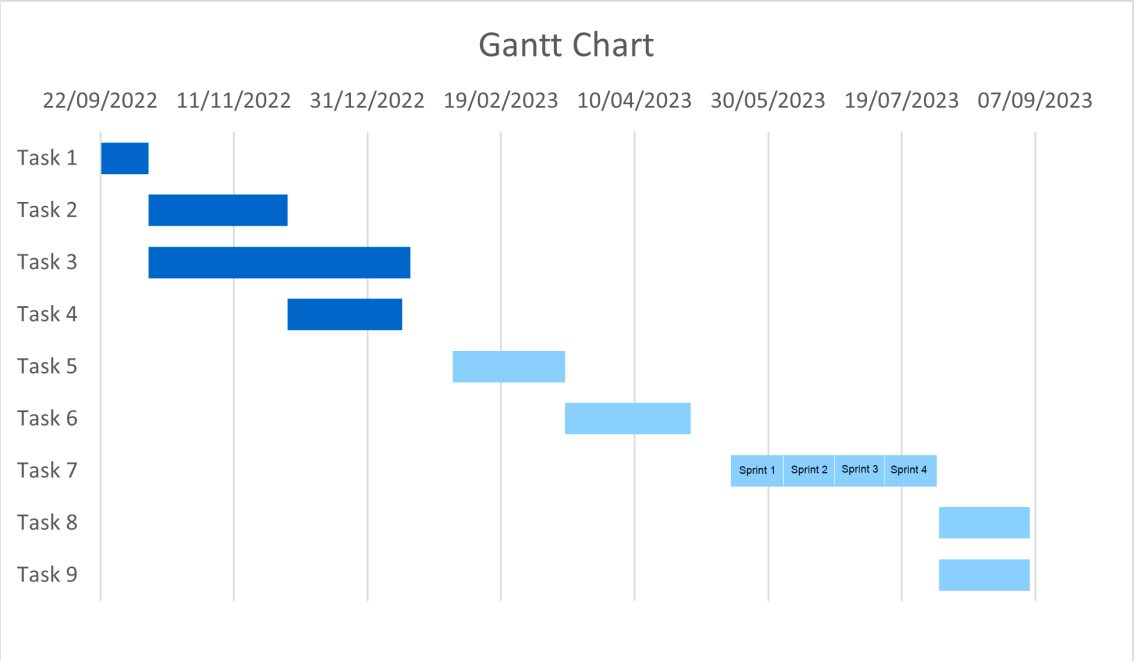


Figure 4.1: Gantt chart with the chronogram of the tasks of the work plan

As we can see by analyzing the diagram, in the first semester, the tasks that took more time to be accomplished was the study of useful subjects related to the objectives of the project, where a study was conducted by reading various articles, where other workers explained their researches, and where we could learn how could we develop the solution for this project. Also, the implementation took some time because we had to create the architecture of the possible solution before implementing it. An analysis was conducted to research possible tools and resources we could use to develop the solution.

In the second semester, as we can observe, a lot of time was invested in the planning phase consisting of tasks 5 and 6, which are better detailed in chapter **??**. To avoid wasting time on changes to the solution and consequently delaying the development of the project, we decided to dedicate a big portion of time to calculating and planning what we were going to do, trying to predict the adversities, and designing the possible architecture of the solution. The design of the mockups, where we discussed through various meetings what the appearance of the interfaces should be like. Finally, as expected, the implementation was the one that took the most time due to the project's complexity, which generated a lot of discussion in the development phase, better explained in section 6.2. It was performed in 4 different sprints, each one with a duration of three weeks. At the beginning of each sprint, we defined a set of tasks to implement in the project and a state of the project wanted at the end of the sprint. Each one of these sprints is better described in section 6.2. The last month was dedicated to writing the dissertation and performing evaluation tests.

4.3 Risk Plan

This section presents a risk mitigation plan, presented in Table 4.2, which shows the possible risks that must be taken into account in the development of the project, and also a mitigation plan to decrease the chances of a risk occurring or decrease the impact of the risk if it occurs.

ID	Description	Impact	Mitigation plan
1	Scope creep: The progress of work suffering a deviation from the goal or objectives	High	- Create clear project parameters from the start - Verify that everyone involved knows the objectives of the project
2	Time risk: The risk of tasks taking more time than scheduled	Medium	- Overestimate the time needed to complete tasks - Overvalue planning, organization, and cooperation with the team to avoid delays
3	Communication risk: The lack of communication means or effort, between the project stakeholders and team members, resulting in loss of data or misinformation, leading to project disruption	High	- Create well defined communication channels, between the team, and with the stakeholders. - Have regular meetings to verify if everyone is on the same page
4	Operational risk: The stallment on project's evolution due to ineffective or failed internal processes, such as people, systems or external events	High	- Have regular meetings to keep up with every members' progress - Do regular backups of the work, so that it does not get lost, due to external factors
5	Performance risk: The project is unlikely to achieve the performance results as intended	Medium	- Study and implement the most suitable solution based on the project goals - Make regular tests to the solution performance, to analyze and conclude if there is a need to change

Table 4.2: Risk mitigation plan

Chapter 5

Design

This chapter focuses on detailing the design aspects of the interfaces that were developed, the system's architecture, and the solution proposal for the Automatic User Interface Generation (AUIG) model.

5.1 Interface Design

Throughout the planning phase, every meeting served as a way to brainstorm new ideas for the interfaces in the visual and functional aspects. The mockups were also presented to the client, where he also contributed to the final aspect of the interfaces.

At the beginning of the planning phase, the functional requirements were carefully analyzed in order to draw the desired sketches. These sketches were drawn with the help of the Figma tool to understand better what we wanted and make it easier to manipulate and change the sketches when needed. The mockups were based on the functional requirements and a document supplied by the client containing a real-life walk-through with detailed descriptions for every action performed from the notification of a maritime pollution event until the end of the combat. With all this information, the objective was to create mockups for every interface needed to simulate this scenario in-game.

Figure 5.1 presents the total number of mockups drawn during the planning phase. As said earlier, these drawings represent the actions presented in the supplied walkthrough step-by-step from the beginning until the end of the combat operations. Each little square in the figure, represents a possible screen of the game.

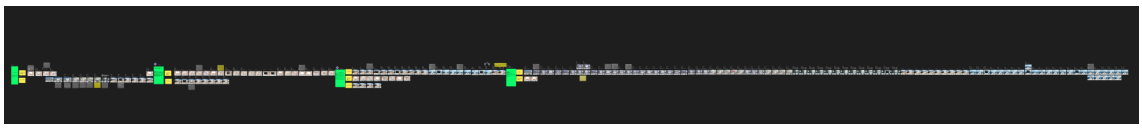


Figure 5.1: Set of all designed mockups

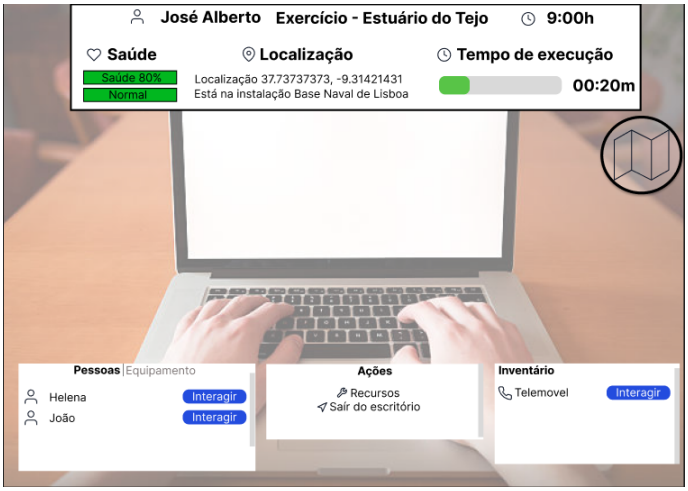
Figure 5.1 serves to highlight the high number of mockups designed and, consequently, the time invested in them. As we can conclude by observing the figure, there are many drawings, meaning that the overall process of combat operations is complex and vast.

5.1.1 Mockups

Below are examples of mockups of some of the core actions of the game and detailed descriptions to explain their functionalities and the decisions made when choosing certain components. These represent some of the core components of the interface, such as the map, and also some of the main actions of the game, such as movement, putting a boom (a maritime barrier that surrounds the spilled oil to prevent it from spreading), interaction with equipment or participants.

5.1.1.1 Main Components of the interface

(a) Main components of the interface, first version



(b) Main components of the interface visually updated

Figure 5.2: Main Components of the interface

Figure 5.2 contains two mockups that explain the interface's main components that are displayed throughout the game's entire time. Figure 5.2a shows the first version of these components. As this is not a 3D game, the background image changes according to the location to give context to the participant of where he is at. The interface contains three different containers: the lists container, the actions container, and the user status. In the lists container, the participant can access the list of close participants to interact with them, as shown in Figure 5.4b; the close equipment list where the participant can interact with the list of close equipment as shown in 5.4a; and the inventory where the participant can access to the equipment that he brings with him. In the actions container, the participant can access the available actions to perform. This container is context-sensitive, which means the actions may change according to the role and location of the participant. The user status contains the participant's status, like his location, health, and indication if he's performing any particular action like eating, resting, or moving. Finally, there is the map button on the left, where the participant can access the map at any time of the game further explained in Figure 5.3.

Figure 5.2b contains the final version of the mockup for the components defined before. The change was due to the visual aspect and other functional aspects, such as the unnecessary step to access the lists or the inventory. As this information is important, this step was removed, and instead, they are always showing so that the participant doesn't have to click every time he wants to check the information from those lists. The status container moved to the top of the page and got a visual update to present the information better, making it clearer. This mockup was only created after finishing every mockup, so the rest of the mockups presented in this section have the first version of the components.

5.1.1.2 Map

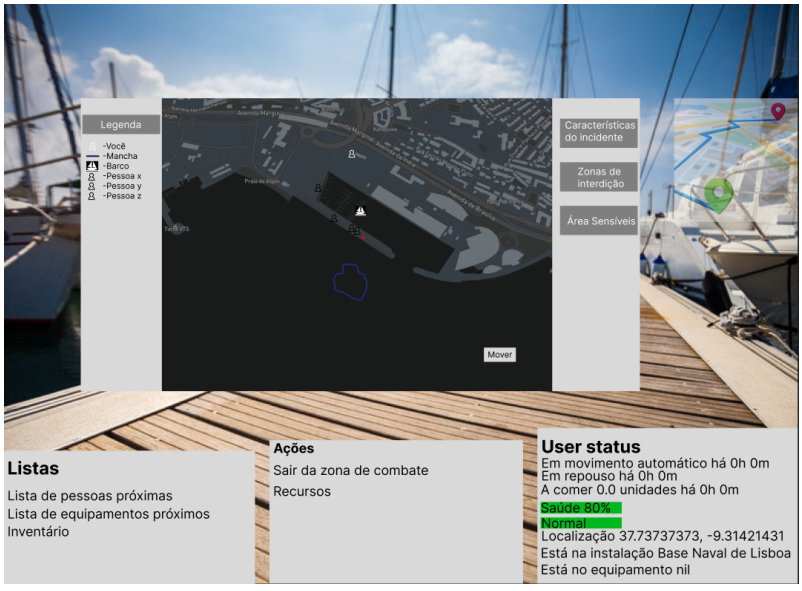
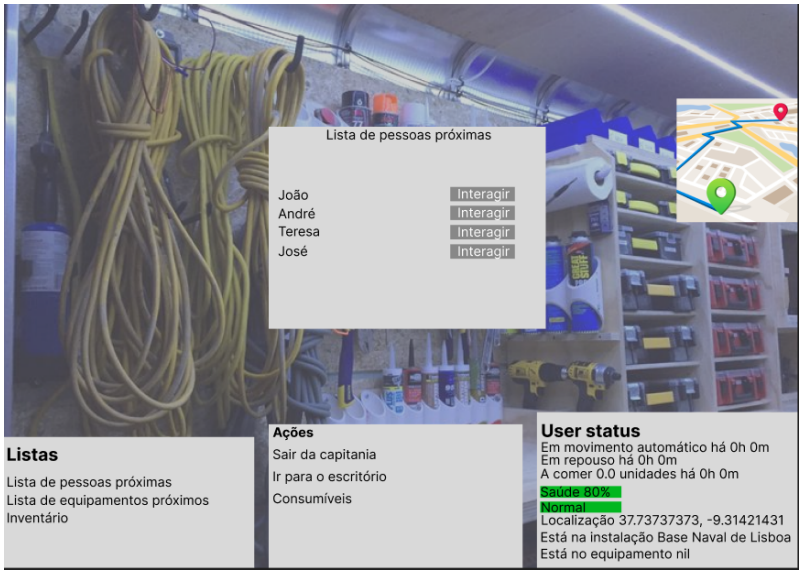


Figure 5.3: Map

Figure 5.3 presents the map the participant can access at any time during the game. This map contains vital information for the course of the game. It contains icons representing other participants, equipments, the spilled oil, and facilities. With this information, the participant is able to guide himself and be aware of the position of the other participants/equipments in the game. In the left part of this container is a legend indicating the entities marked on the map and their name. On the right part, there are three buttons that, when clicked, apply a layer to the map, such as the interdiction zones or the sensible areas. The other button allows the participant to get the characteristics of the incident, such as the volume of spilled oil, the number of injured people, and other characteristics. These layers are explained better in section 6.3.

5.1.1.3 Close Participant/Equipment List

(a) Close Equipment List

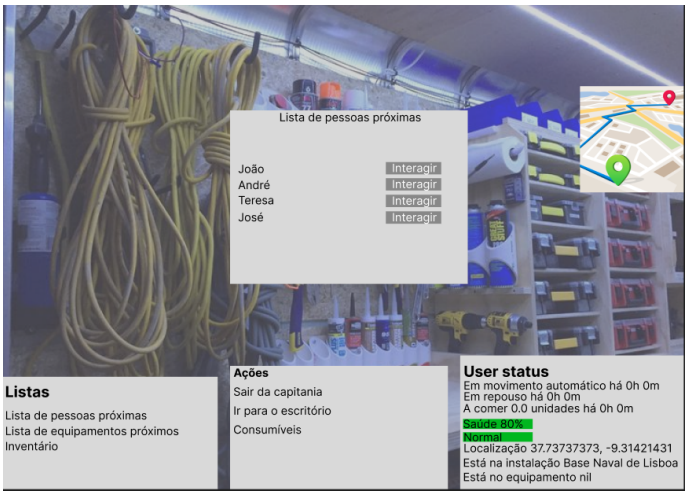


(b) Close participant List

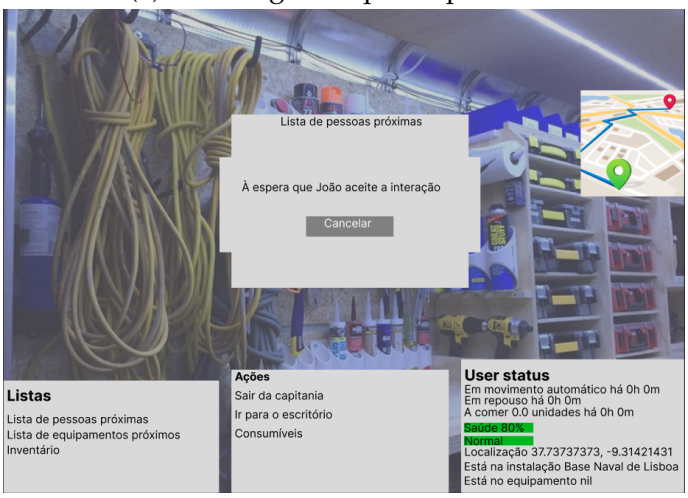
Figure 5.4: Participants and equipments close by

Figure 5.4 presents the result of clicking in the participant/equipment close lists. It shows an interface with the list, where each element has a button to interact with the element. In the case of trying to interact with a participant, there is a try to live chat with him. This is better demonstrated in Figure 5.5. When trying to interact with an equipment, an interface appears, showing the status of the equipment and actions to perform with it, as it is better shown in Figure 5.6. As said earlier, these interfaces were not implemented because this information is constantly showing on the screen instead, but they are being presented to give context to other mockups.

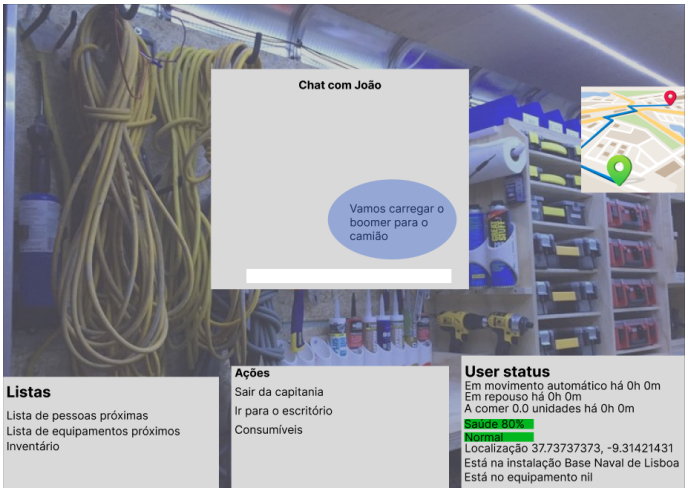
5.1.1.4 Interaction with other participants



(a) Acessing close participant List



(b) Waiting for the other participant to join the chat



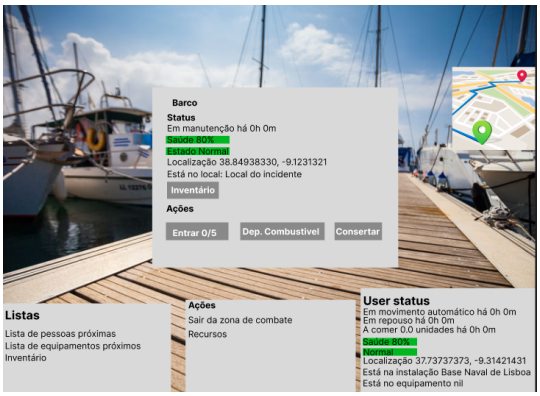
(c) Chat with other participant

Figure 5.5: Interaction with a participant

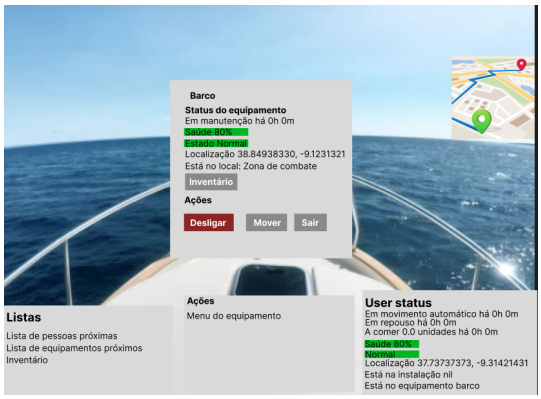
Figure 5.5 presents the process of interacting with a participant. When pressing the button interact, the participant waits until the other participant connects, and after, they can write messages in the chat to each other. When implementing the communication between participants, there were some changes that are better explained in section 6.3, which shows how the communication was implemented and the differences in relation to this mockup.

5.1.1.5 Interaction with a vehicle and participant movement

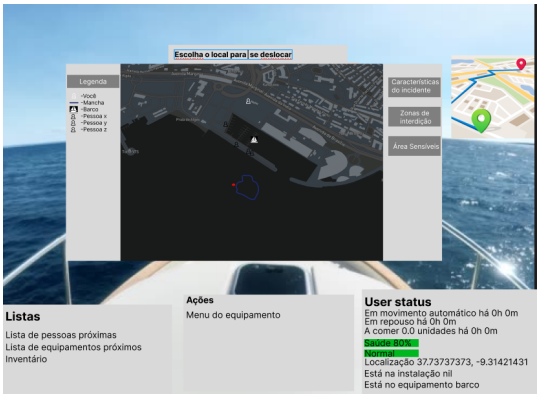
(a) Acessing close equipment List



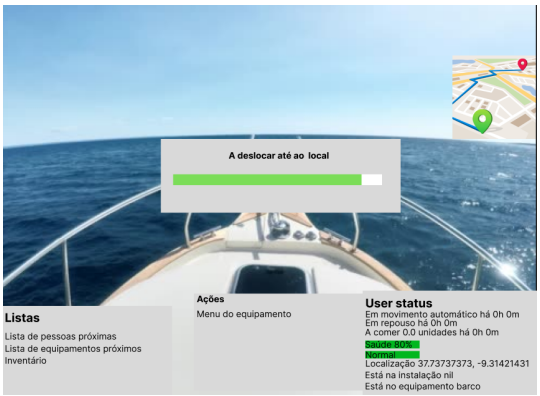
(b) Boat Interface outside the boat



(c) Boat Interface inside the boat



(d) Choosing the point in the map to where to move

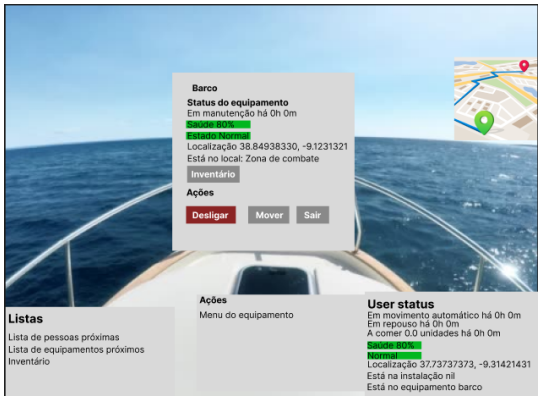


(e) Movement Interface

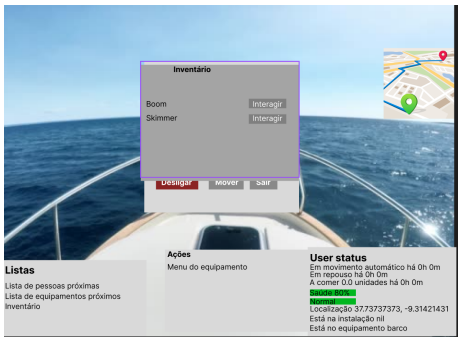
Figure 5.6: Interaction with equipment and movement

Figure 5.6 presents the interaction with a vehicle. This interface component contains the status of the vehicle and the possible actions to perform with it in the current context. In order to drive it and move, we need to access the list of close equipment and interact with the boat to get the corresponding interface, as shown in Figure 5.6a. After entering the vehicle, we have the move action, which, when clicked, shows the map. Then, the participant only has to click on the destination point, and an interface showing the action being performed appears.

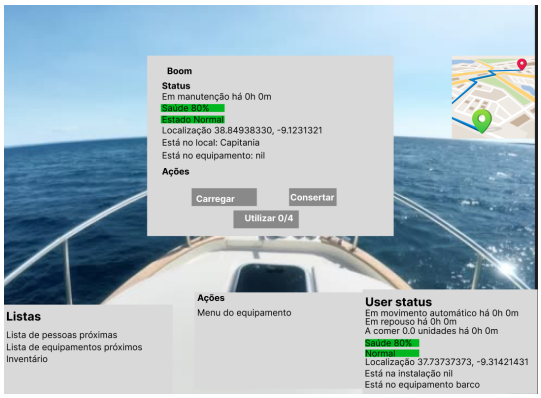
5.1.1.6 Placing a boom



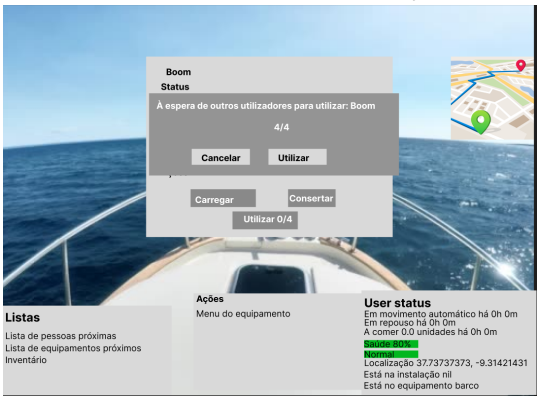
(a) Boat Interface



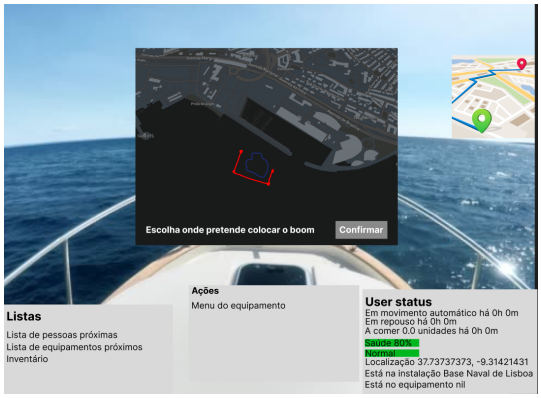
(b) Boat Inventory



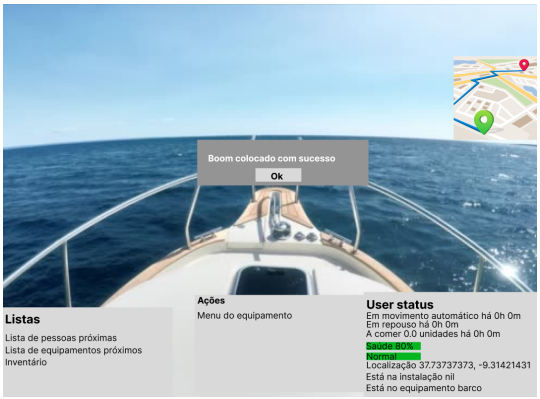
(c) Boom Interface



(d) Waiting for people to help put the boom



(e) Chosing the points in the map to put the boom



(f) Message of success after putting the boom in water

Figure 5.7: Placing a boom around the spilled oi

Figure 5.7 presents the action of putting a boom in the water to contain the spill. To do so, we need to be inside a boat, next to the spill, and have a boom in the boat's inventory, as shown in Figure 5.7c. This action requires more focus on choosing the points for the boom to be put in place. As shown in Figure 5.7f, the participant chooses the points of the vertices of the boom to be put by clicking on the map until he hits the confirm button to end the action. The objective of this

action, regarding the interaction, was to have an interactive moment so that there is more precision when choosing the boom's vertices and that the game does not get monotonous with only button clicks.

These were examples of mockups of some of the most important actions in the game. In section 6.3, the corresponding interfaces for these actions are shown again, explaining every change that happened, implementation adversities, and interface improvements.

5.2 System Architecture and Interface Generation

This section presents the system's architecture and the model for the interface generation.

5.2.1 System Architecture

As mentioned in chapter 1, these project is divided in five different dissertations, each with different objectives and tasks, set to develop a component to be integrated with the other components and form the game. The cooperation and coordination between these components was mandatory for the success of the project, hence the need for the system's architecture to be robust and well designed. Let's remember each general task of each dissertation's component:

- **Digital twin** - Works with a mathematical model system called MOHID responsible for giving realism to the game by calculating, for example, the increase/decrease in the spilled oil taking. This component aims to be the intermediary between MOHID and the other components.
- **Game editor** - is responsible for creating the data model of the game, like the entities and relationships, for example.
- **Multiplayer Service Framework** - Is in charge of creating the game's multiplayer service and establishing the communication channels between every component.
- **Automatic User Interface Generation** - Responsible for automatically generating every interface based on the game's current state.
- **VR Operations** - where It is concerned to implement a virtual reality interface for specific actions in the simulator.

Although there are five components, the project was done on the first four because it was not possible to integrate the VR operations together with the rest of the components. Below is represented the System's Architecture.

Figure 5.8: System's Architecture

Figure 5.8 represents the System's Architecture. The four components are represented, plus the database and MOHID. As we can observe in the architecture, this dissertation's component is the intermediary between the client (browser) and the server. It communicates with the use of WebSockets or controllers (Spring boot), which will be better explained in chapter 6. The multiplayer Server framework is connected to every other component because it is in charge of updating the state of the game by passing the information to every component. As mentioned earlier, the Digital Twin component is the intermediary between the other components and the MOHID. Finally, the Game Editor component is in charge of creating the entities and methods to create or alter the state of the repository.

When an action happens in the browser (any type of request), we can create a set of steps of what happens then:

1. The UI generation component communicates with the multiplayer component, indicating the request.
2. Then the multiplayer service framework, based on the request, makes the necessary communications to process the request, which can be:
    - Communicates directly with the database or uses a method from the Game Editor component to alter the state of an entity in the repository, for example.
    - Communicates with the Digital Twin in case the request needs to be processed by MOHID, for example, updating the state of the spilled oil after putting a boom or a skimmer.
3. With the result, the multiplayer server framework communicates back the reply to the UI generation component
4. Lastly, the UI Generation component makes the necessary changes to the interface in the client, based on the received reply.

Below is represented the entity-relationship diagram:

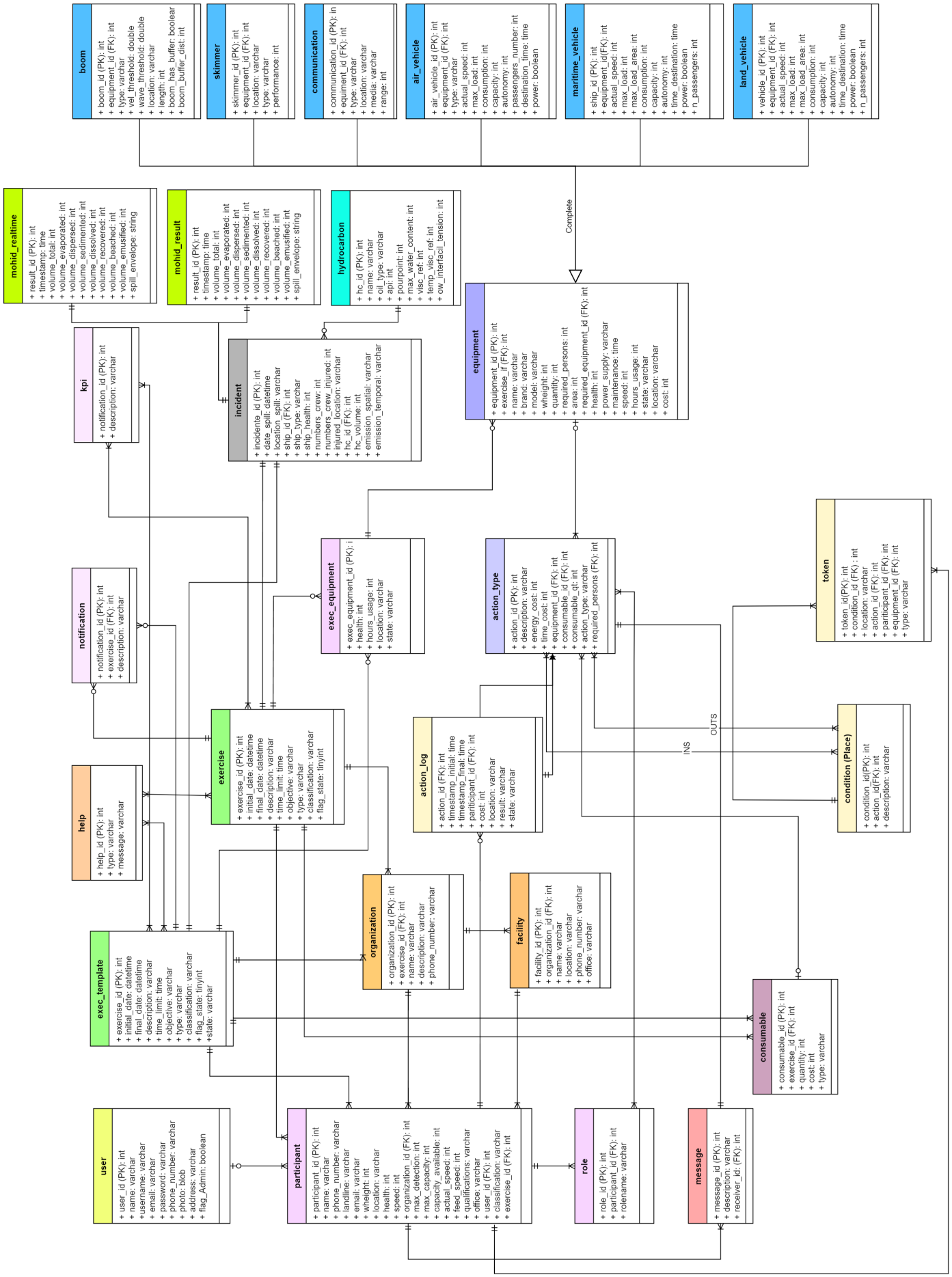


Figure 5.9: Entity-Relationship Diagram

This project is complex and vast, and in order to meet the supplied functional requirements and the project's goals, there is a need to have a solid and robust data model, as represented in Figure 5.9. As we can observe, this diagram is complex. It has a lot of entities and relations between them. The following are the most important:

- **Exercise** - An exercise mainly consists of an incident (maritime pollution event), participants, equipment, and organizations. The exercise is the game itself where the users are going to play at. The exercises can be configured to alter the data of the aforementioned entities in order to create realism and diversity.
- **Users and Participants** - Users are real people and Log in to the website of the project. When it comes to playing the game, they are associated to a participant, that is, a fictitious person that has a role in the game. Participants have roles, are associated to organizations, and can have equipment, and consumables with them.
- **Organizations** - Organizations represent themselves in the game. They consist of facilities that have equipment, consumables, and participants associated with them.
- **Incident** - Incident consists of the event that triggers the beginning of the exercise. It is associated with a type of Hydrocarbon and other important information relevant to the game.
- **Messages and Notifications** - The way of communication between the participants in the game is represented by the entity *message*. The notifications are shown to the participants when they receive a message from other participants or there is some update in the game that requires the participant to be noticed.
- **Equipment** - The equipment has various types: vehicles, like boats, cars, airplanes; Skimmer; Boom or other pieces of equipment that can be transported by the participant, like phones, for example. They are associated with actions that the participant can perform with each piece of equipment.

With the entities' diagram, together with the system's architecture diagram, we managed to have a solid base plan to start developing the initial state of the project, like the database, the communication channels between each component, the infrastructure, and the components of each dissertation.

5.2.2 Interface Generation

Interface generation was achieved with the use of templates. Out of all the approaches studied, this seemed to be the most appropriate. By studying these approaches, we have obtained important information and ideas to implement in this dissertation's project. But, for some of the approaches it would be necessary to use external systems to use them, such as SUPPLE [10] and TERESA [3]. Since it will already be difficult to coordinate all the components of the Marine Polution Control Simulator (MPCS) project, we wanted the approach used to be easy to integrate to maximize decoupling. Using templates, we are able to implement the automatic user interface model using the technologies and languages that the other components of the MPCS project will be using. In addition, other approaches studied required a study of the technologies and tools to be used to accomplish the automatic generation, such as PUC [9], where using templates, the needed study is way less. We only design a model of how the generation will work, and start implementing right away, as the usage of templates is simple. Finally, after studying and analyzing the possible the mockups, we reached the conclusion that only a few interfaces should be dynamically generated since there are a lot of interfaces that will be static without changing their state, like menus, and others that can change the state but are fixed in the screen, like the profile of the participant, the lists of close participants/equipment and the inventory; therefore, they were generated manually. We reached the conclusion that the interfaces that need to be generated automatically, can easily get grouped where a single template can generate various different interfaces maximizing efficiency and decreasing development time.

Therefore, these reasons made the use of templates the choice for automatically generating interfaces.

The mockups were studied and grouped to create templates that would work for different occasions to maximize efficiency and decrease development effort. Also, it was conducted an analysis on what data should be in each template and what data should be irrelevant. In the case of equipment/consumables, only the most important data is shown, like the health, location, and other relevant data depending on the type of equipment, like the autonomy for vehicles or the quantity for consumables.

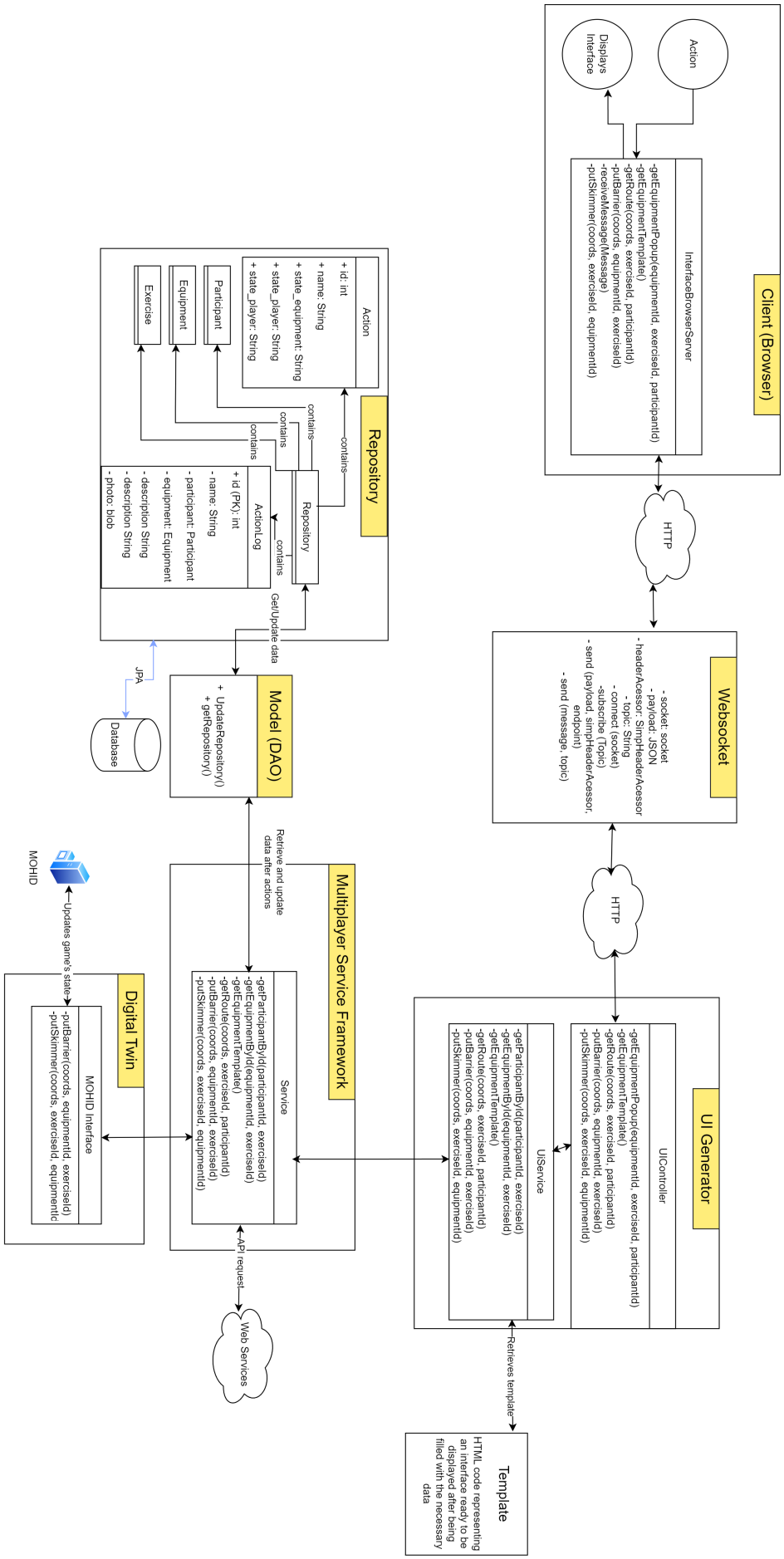


Figure 5.10: Interface Generation Model

Figure 5.10 represents the architecture of the interface generation model using templates. When an action occurs on the browser, for example, putting a barrier/skimmer, asking to interact with a piece of equipment, or wanting to move, it triggers an event that sends a request, using WebSockets, to the respective endpoint to process the request. The information in the request will vary depending on the context of the game. The server contains a controller mapped by this endpoint. This controller calls the necessary services to process the request. These services can: obtain/update information in the database; call a defined method to process the information, for example, a method to calculate the distance between two points; call a method to make an API request to the Digital Twin to exchange information with MOHID and update the game's state; call a method to make an API request to an online service, for example, obtain the GPS route between two points. After finishing processing the information, sends it back to the client. Then, the client requests the corresponding template. This template has empty divisions appropriate for the type of use case they are being generated. Then, the client creates searches for these divisions in the template and fills them it with the corresponding data obtained before. After inputting all the data in the template, displays it.

5.2.2.1 Problems encountered

Although this approach generates the interfaces dynamically based on the context of the game in an abstract way, the templates are already created. This model of generating interfaces is coded brutally to work as it is, to work with the current version of the game. This can generate possible problems in the future because the interface is always the same for the specific situation to be generated, which means that the type of data to be displayed is always the same, and so is the visual appearance of the interface. So, if the game's administrators want to see a new equipment's field displayed on the interface, they can't. Another situation is if the game's administrators want to add a new type of entity to the game and there is no suitable template to generate the interface. Finally, there is no way to customize the visual aspect of the generated interfaces. These problems can be solved by changing the template's code and the system's code, but it's not practical for the administrators to do it, especially because they may not have any experience in coding. Therefore, section 5.3 presents a proposal of a model using AUIG capable of solving these problems, and section 6.3 shows how they were implemented and the results.

5.3 UI generation Model using a JSON configuration file

I propose the following automatic interface generation model that aims to solve the automatic generation of user interfaces using templates by giving administrators complete customization of interfaces and complete abstraction to the generation of interfaces. This model was only applied to the equipment interface

since it's the one that can change the most. This model generates the interfaces using an empty template and a JSON configuration file. The file contains the interface specification for each type of equipment and a default specification in case the type of equipment is not specified in the file. This specification contains the equipment fields that will appear in the interface and also their datatype in a high-level description, such as *text* for strings, *numeric* for numbers, and *range* for numbers that belong to a range of numbers that is also specified. This file doesn't contain details about the visual aspect of the action buttons or the inventory button, so the customization only applies to the equipment fields. Each field has an associated set of properties that will define the visual appearance of the interface. These properties are the following:

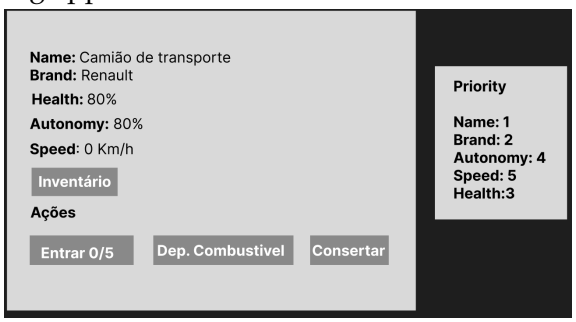
- **Priority**: Defines the order in which the elements appear in the interface. The possible values are numbers. The lower the number, the higher the priority.
- **Interaction**: Defines whether the participant can interact with the field. If the value is *none*, the field is displayed normally. If it's not *none*, it can be *select* to be represented by a component with various options to select the field's value, such as a dropdown. The property value can also be *drag* to be represented by a draggable component, such as a slider. Finally, it can be *upDown* to be represented by a component that can increment/decrement the current field's value. This property is used for fields that can be changed during the interaction to change the state of the equipment, such as the speed of a vehicle or the power of a piece of equipment in use.
- **Grouping**: Defines whether certain fields are close or not. The value can be *none* or a number, and the fields with the same number will be closer in the interface. The Priority property, together with this property, defines the order of the elements in a group.
- **Relevance**: Defines the component's relevance in the interface, whether the information should be more noticeable or not. The possible values are *Low* for subtle indicators of the field, such as less size and more discrete background colors and shadows; *medium* for medium indicators of relevance and *High* for highlighting critical information.
- **Customization**: Defines the color and font of the field. The possible values are *none* to have the default color and font of the game, or it can be *($font, $color)* where *$font* should be replaced by an existing font and *$color* should be replaced by an existing color.

5.3.1 Mockups

To give better context about how the properties should be used, the following figures represent the designed mockups explaining the use of each property.

Priority

(a) First example of the Property Priority being applied



(b) Second example of the Property Priority being applied

Figure 5.10: Property Priority being applied

Figure 5.10 shows two examples of how the property priority affects the interface. Each Figure has the mockup on the left and the corresponding values used in the priority property. As mentioned before, this property sets the order of how the fields should appear in the interface. Figure 5.10a shows the result of the field autonomy being placed first because of its priority having value 1. Then succeeds health, brand, and speed, for having priority values 2,3 and 4, respectively. Lastly is the field name, where its priority is 5. Figure 5.10b shows another example of how this property looks like.

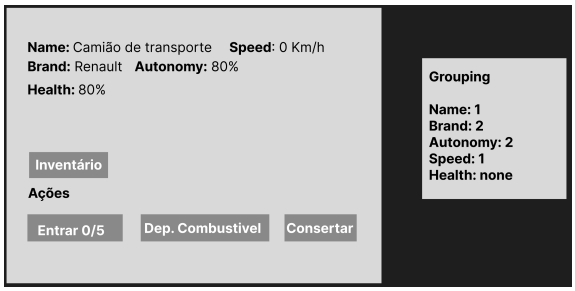
Customization

Figure 5.11: Property Customization being applied

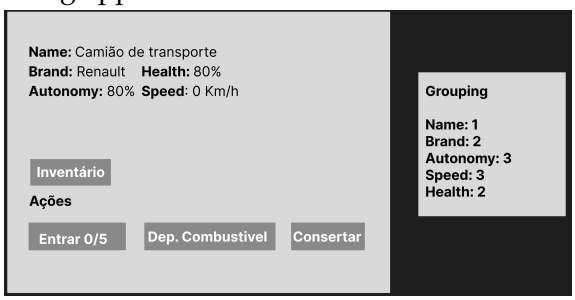
As mentioned before, the customization priority sets the font and color of a field.

As we can see in Figure 5.11, on the left are the values used for each field, and on the right is the corresponding result of using each font and color.

Grouping



(a) First example of the Property Grouping being applied



(b) Second example of the Property Grouping being applied

Figure 5.11: Property Grouping being applied

The grouping property aggregates fields that have the same grouping value. Figure 5.11a shows the corresponding grouping values on the left, and on the right, we can see that name and speed are together because of having grouping value 1. Then brand and autonomy are together because of having grouping value 2. Finally, health is isolated because it has grouping value *none*. Figure 5.11b shows another example of how this property works

Interaction

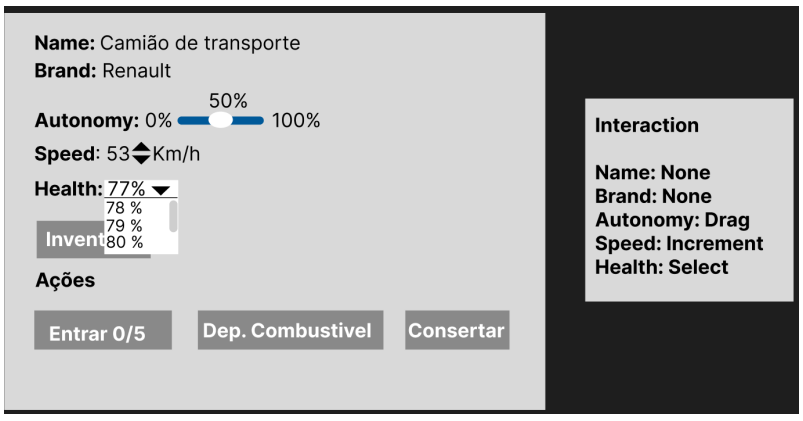


Figure 5.12: Property Interaction being applied

The interaction property defines if a field can be interactable or not. Figure 5.12 shows that name and brand are displayed normally, while autonomy being displayed as a draggable slider because of having Interaction value *Drag*, speed being displayed as an increment number because of having interaction value *Increment* and the health being displayed as a dropdown because of having interaction value *Select*.

Relevance

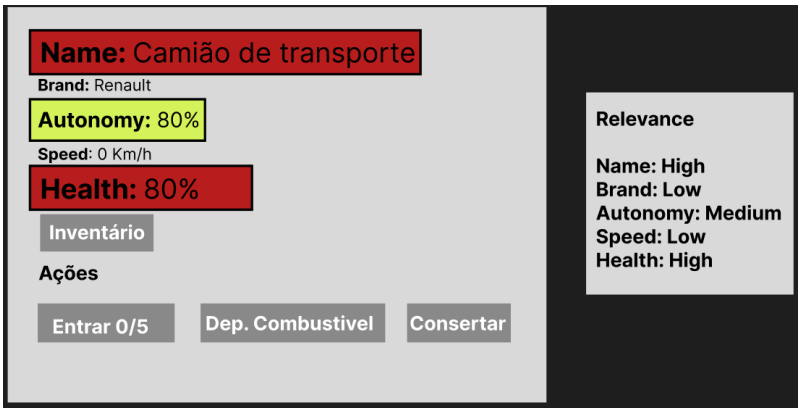


Figure 5.13: Property Relevance being applied

The relevance property defines how much attention a field should get. Figure 5.13 shows that fields that have a relevance value *High* have more accentuated borders, more critical colors, and bigger sizes. The fields with relevance value *Medium* are also highlighted but slightly less than the aforementioned. Finally, the fields with relevance value *Low* are the ones that are less noticeable in the interface.

All these properties define the visual aspect of the interface to be so that the users can decide on the aspect and change at will. The properties are customized directly on the JSON file. This, together with the equipment data, should be enough to generate interfaces on an abstract and customizable way.

5.3.2 Architecture

The idea of this model is to generate an interface only based on the equipment's data and the JSON configuration file. Figure 5.14 shows the diagram that presents how this model works.

Figure 5.14: UI generation Architecture

As we can see by observing Figure 5.14, the process of generating and interface is similar with one using templates. However, in this model there is acess to the JSON configuration file to help generate the interface.

When an action occurs in the client, there is a method call that makes a request, using WebSockets, for an interface to the server. When the Spring Boot controller receives the request, makes a request to the database to obtain the equipment data, the participant making the request and, based on this information, gets the possible actions to perform. Then, with the equipment, it calls a method that reads the JSON file and returns a JSON object containing the fields that should appear in the interface plus their properties. This information is replied to the client. Then, the client makes a request for a template, where the Spring Boot Controller searches for it and replies it. The client then fills the template with the information received and presents it to the user.

This model can generate the interfaces in an abstract way, depending on the current context of the game, and also has the customization aspect for the administrator to define how he wants the interface to be like. With this being said, this model is expected to solve the problems of interface generation using templates. In section 6.3.2, the implementation of this model is further detailed, as well as the results.

Chapter 6

Development

This chapter focuses on the development phase of the MPCSproject and this dissertation's project. It presents the development details of the infrastructure, the chosen technologies, the development process describing the achievements in each sprint, and the final practical results of the automatic user interface generation using templates and the the automatic user interface generation using a JSON configuration file.

6.1 Infrastructure

This section presents the details of why Spring Boot was chosen for the server-side framework, HTML for the client-side interface, and MySQL for the database management system, drawing from each development team member and highlighting each choice's advantages.

6.1.1 Choice of Technologies

Spring Boot

The decision to utilize Spring Boot as the server-side framework was primarily driven by the experience with this technology. Spring Boot is a well-established and widely adopted framework in the industry. It offers several advantages that make it a suitable choice for this project [14]:

- **Rapid Development**: Spring Boot simplifies the development process by providing a comprehensive set of libraries and tools. Its convention-over-configuration approach reduces the need for boilerplate code, allowing developers to focus on the application's core functionality. This accelerates development and ensures code quality.
- **Robust Ecosystem**: The Spring ecosystem provides a wide range of modules and extensions that seamlessly integrate with Spring Boot. This allows

for the easy implementation of various features such as security, data access, and messaging.

- **Strong Community Support**: Spring Boot boasts a large and active community of developers, which means access to extensive documentation, forums, and resources. This support network can be invaluable when encountering challenges during the project.

HTML

HTML was chosen for the client-side interface due to its simplicity, ubiquity, and familiarity with it. While HTML itself is not a programming language, it forms the backbone of web development and offers several advantages for creating user interfaces [15]:

- **Cross-Platform Compatibility**: HTML is universally supported by web browsers, making it an excellent choice for creating a client-side interface that can be accessed from a wide range of devices and platforms.
- **Lightweight**: HTML is lightweight and efficient, ensuring fast page loading times and a smooth user experience. This is crucial for web applications, as user satisfaction often hinges on responsive interfaces.
- **Ease of Learning**: HTML is relatively easy to learn, making it accessible to developers.

MySQL

The selection of MySQL as the database management system was influenced by previous experience and its suitability for the project's requirements. MySQL offers several advantages for managing relational data:

- **Data Integrity**: MySQL is renowned for its robust data integrity features, including support for ACID (Atomicity, Consistency, Isolation, Durability) transactions. This ensures the reliability and consistency of the data stored in the database.
- **Scalability**: MySQL can efficiently handle large volumes of data, making it suitable for projects that may experience growth in the future. Its scalability options, such as replication and clustering, provide flexibility in accommodating increased workloads.
- **Cost-Effectiveness**: MySQL is open-source, which means it is cost-effective and aligns with budget constraints. This is especially valuable for academic and research projects where cost efficiency is a consideration.

In conclusion, the choice of Spring Boot for the server, HTML for the client, and MySQL for the database was based on a combination of previous experience and the inherent advantages these technologies offer. Leveraging familiarity with

these tools allowed for a more efficient development process and a higher likelihood of project success. Additionally, the selected technologies are well-suited to the project's requirements, ensuring robustness, cross-platform compatibility, and cost-effectiveness.

6.1.2 Server deployment

The project was run most of the time on localhost. Each team member carried out the tasks separately or together, depending on the task. To keep the project up to date and coordinated, we used GitHub to keep up to date with the progress of the other members and to coordinate our work better together. At the end of the development phase, we deployed the project on a server provided by the Department of Informatics Engineering (DEI), so that the project could run online to carry out some tests with several players simultaneously and finally get the project ready to show to the client.

6.1.3 Client-Server communication channels

The Client-Server communication is the core aspect of our system since every action in the game makes a request to alter the state of the game and receives back the response. Everything is processed in the server, and the client only shows the game's current state. Since we used HTMl as the client language and because HTML is static, the usage of WebSockets and Spring Boot controllers creates the capability of having the interfaces change dynamically during the execution of the game.

WebSockets

When a player connects to the game, two WebSockets topics are created, one where he only receives messages for himself and has the endpoint with the following format: */exerciseId/participantId*. This topic receives messages such as updating his life bar or getting equipment information. In the other topic, he receives messages destined for all participants and has the endpoint with the following format: */exerciseId*. This topic receives messages such as, for example, when a participant moves, a message is sent to all participants, which serves to update each participant's map with the new location of the moving player. These topics receive messages throughout the game to update the game interface. To distinguish each message received, they contain a label that defines what the client does when receiving such a message. Then, depending on the label, an If block separates the messages to execute different blocks of code. The message also contains the necessary data to make the changes to the interface. Below are some examples of the labels and messages used:

- **UPDATEHEALTH**: When a message with this label is received on a topic, it also contains the new health value. The code enters the if statement when the message label equals this one and calls a function that updates the health

element in the interface with the new value. This message is normally received when a participant's health suffers an update.

- **CLOSEPARTICIPANTSANDEQUIPMENTS**: When a message with this label is received on a topic, it also contains a list of the name, id and location of the participants and equipment that are close to the respective participant. Then, it calls a function to update the close participant list and the close equipment list with the new received. This message is when a player/equipment moves, and there is a need to update the corresponding lists.

- **PARTICIPANTSANDEQUIPMENTS**: When a message with this label is received on a topic, it also contains a list of the name, id, and location of every participant and equipment in the game so that the map's icons and legend can get updated. This message is normally received when there is a need to update the map, either by someone's movement or action that requires an update to the map.

- **CHAT**: When a message with this label is received on a topic, it also contains: the type of chat received, whether it is an email, message, or someone spoke; the content of the message and the sender. A function is called to display a notification of the message received.

This is how the WebSockets deal with the messages received in topics. However, they can work the other way around. They can perform a request to an endpoint mapped by a controller, as it was mentioned before. Depending on the action that requires the request, a message is built and sent to the respective endpoint, where the controller processes the message and replies to one of the player's topics.

Spring Boot Controllers

Spring Boot Controllers are classes with the annotation *@Controller* that have methods that map an endpoint to execute the respective method when accessed to this endpoint. These methods can return templates to show them in the interface or can have the type void and return nothing. For example, when we wanted to show a popup or change the page, we made a request to one of these methods, and he would return the corresponding template to show. When we didn't want to change the page but only interface components, the return type of the method would be void, and we created Objects of the class *SimpMessagingTemplate* to send information to a WebSocket Topic to make the necessary changes, as it was explained earlier. Before returning the new template or sending a message to a WebSocket topic, we would call services or methods to process the request received in the endpoint. Below is a diagram showing better how the communication works and its effect on the interface.

Communication Example Diagram

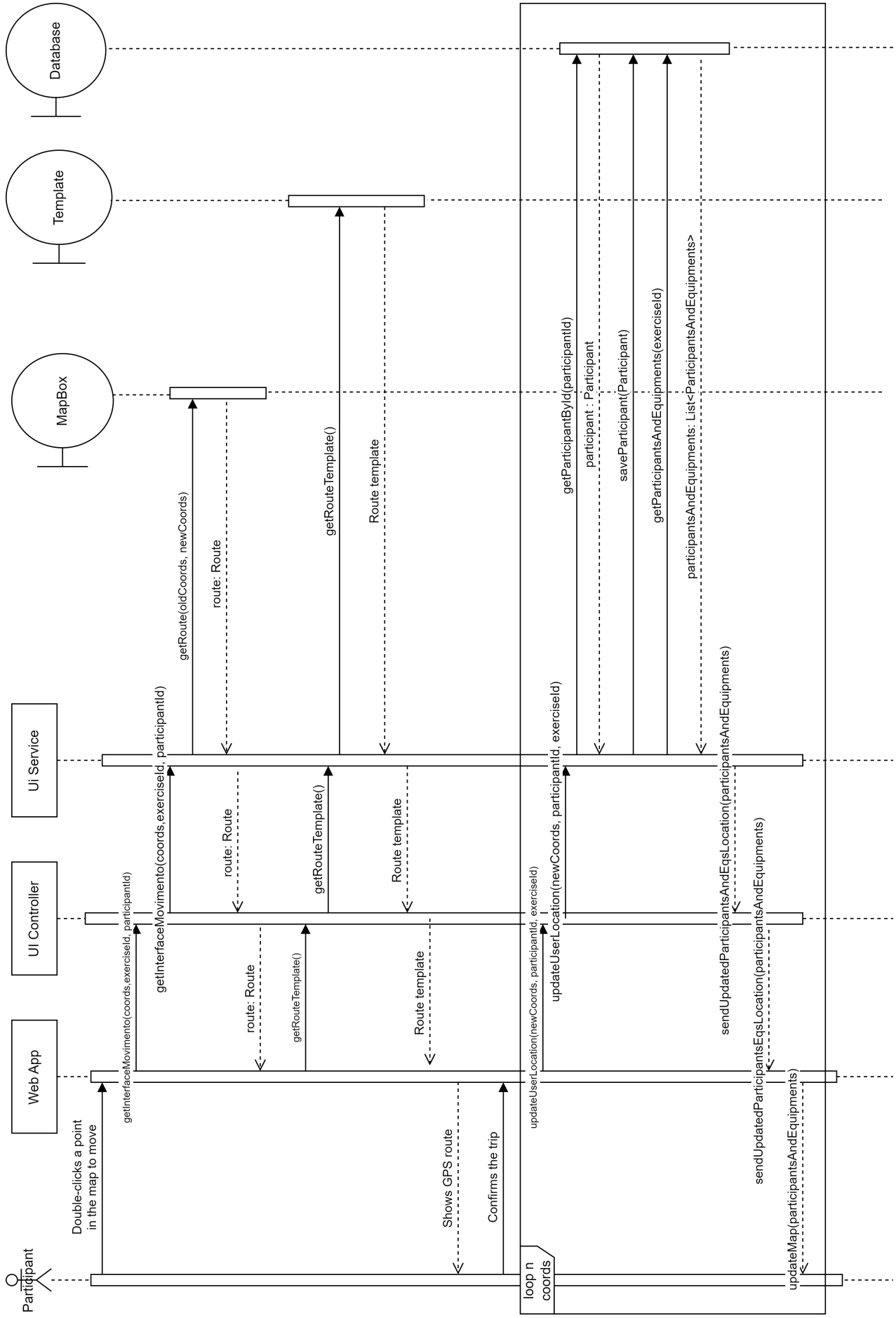


Figure 6.1: Diagram showing the communications made for a player to move

Figure 6.1 shows the exchange of information when a player wants to move. When he double-clicks the map, a request is sent to the endpoint */interfaceMovimento*, where an API request is made to obtain the GPS route information. After obtaining the information, reply with the full information about the route to the destiny, such as the coordinates to travel, the distance, and the duration to the participant's WebSocket topic. With this information, the client makes another request to the endpoint */routeTemplate* to get the appropriate template to fill with the obtained information. This interface is presented in Figure 6.6. This interface aims to show the participant the route before moving so he can confirm or decline the movement. After confirming, the function *simulateMovement* is executed on a loop, having a time interval between each loop. The time interval depends on the duration and number of coordinates to travel. This function updates the participant's location on his map and sends a request to the server to update his location in the database. Then, a message is sent to every participant's Websocket topic with the updated list of participant equipment names and their location and the updated list of close of participants and equipment. Upon receiving this information, the client updates every participant's map and their close lists of participants and equipment. This process simulates the participant's movement and displays it on everyone's map. This means that if the other participants have the map opened, they can see the participant with the color yellow, which indicates that he is moving. Also, they can see the participant's marker moving gradually through his corresponding route.

This diagram shows how the interfaces are updated dynamically and how the Client-Server communication works. Every in-Game-Action works similarly with this, meaning the client receives messages in the subscribed topics and executes different things according to each message; and sends messages using WebSockets that Spring Boot Controllers receive in the server, which process the request and reply to one of those topics. This way, we can keep the client always in synch with the server.

6.2 Development process

The Development Phase was divided into four sprints of three weeks. In each sprint, the development team members had tasks assigned. During each week, there were meetings where the team members showed the progress and the adversities. These meetings served to discuss solutions, alterations to make, and new ideas to implement. Close to the end of the sprint, there was a testing phase where we tested the implemented functionalities. In the meeting where every sprint ended, there was a final presentation about the progress, a planning phase, and a new assignment of tasks for the next sprint.

6.2.1 Sprint 1

The first Sprint was dedicated to setting up the project and starting to build the infrastructure. It consisted of creating the database and building the project and

its connections to the database using the JDBC API methods, which consisted of the methods portraying the operations reading/deleting/updating/inserting with the database. Then, we proceeded to develop the infrastructure with small increments by creating small development demonstrations/scenarios, where we could test and implement simple functionalities in the project, such as retrieving every user in the database and showing them in the interface to test the use of the Spring Boot Controllers and WebSockets and their configuration. Then, we created an exercise where the users could join. When they joined, they had a chat where only the users from that exercise could talk with each other. Both scenarios simulated the communication between the Multiplayer Service Framework and Game Editor components using WebSockets, Spring Boot Controllers, the database, and a message queue that we thought was needed at that moment for the communication between the components in the server. Also, the interfaces used for the scenarios were static and manually developed because the purpose was to establish and set up the project first. This initial prototype established the groundwork for the project to evolve into the simulator. At the end of the sprint, we planned for the next sprint to start implementing in-game functionalities.

6.2.2 Sprint 2

The second sprint was dedicated to starting the implementation of the various functionalities of the game. Throughout the implementation, we realized we didn't need a message queue to communicate between the server's components. Instead, we only needed to do method calls from the different components. The communication using WebSockets and Spring Boot Controllers stayed the same. After some discussion, we concluded that the components of the interface of the profile status, the close list of equipment and participants, and the inventory should always stay visible and, therefore, manually created. However, they should be dynamically updated with the current state of the game with the help of the established communication channels. The rest of the interfaces would already be developed using templates. In this sprint, the following actions were implemented:

- **Fixed Interfaces**: The fixed interfaces are the ones that are always visible to the participant and only change their state dynamically. These include the participant's profile status interface, the inventory, and the close lists of participants and equipment. In the planning phase, the mockups also had a component with the possible actions, but after some discussion, this component was not implemented as the participant can perform every action anyway by using other components such as the map or the equipment/consumable interfaces. These interfaces, in terms of functionalities, didn't change. However, throughout the development phase, the visual appearance changed a little.

- **Map**: The map was implemented using the leaflet library. The start of using this library was difficult because there was no prior experience with anything similar, such as drawing figures in maps, placing markers, and the

usage of coordinates. However, there is a lot of material online to help the development of leaflet maps. With the map implemented, we were able to place markers indicating participants/equipment/facilities in their respective locations so that the participant could guide himself better when playing the game. The map is also used to move. Firstly, the map container had a button move, where the player clicked so he could choose a point in the map to move to. After discussion, we didn't find that visual intuitive or appellative, so that button was removed, and, instead, the participant can move using a double-click on the map. There were also implemented verifications in case a player wants to move/drive to water or a boat to land. These verifications were implemented using API Services that verify if a coordinate is on land or water.

- **Equipment/Consumable Interface**: This is where Automatica User Interface is more noticeable using templates because there are a lot of different equipment/vehicles/consumables to generate an interface for. When a participant interacts with an equipment/consumable close by, an interface appears showing the details and status of that equipment/consumable. In this Sprint, it was only possible to implement this informative interaction, not being able to do some action with them.

- **Sensible/Interdict Areas**: These areas are important for the game's punctuation after the ending of the game. They serve to verify if a participant has made an action in an area he shouldn't do. The participants draw These areas on the map at the beginning of the game and color it according to their classification (green zone, yellow zone, red zone). They can be accessed using the layers that a player can see when choosing the filters on the map container. These areas are drawn polygons in the map. There were some adversities again because there was no experience working with drawing in maps, but in the end, they stayed pretty much the same.

For the next Sprint, we planned to implement more functionalities and have such functionalities that could simulate a pollution event and carry out combat against it.

6.2.3 Sprint 3

This sprint aimed to implement enough functionalities/in-game actions to simulate an exercise of combat to a pollution event. The following functionalities were implemented:

- **Spilled oil**: With the coordinates of the pollution event and its spilled oil, we draw a polygon in the map. This is done using GeoJson, that is having the characteristics of polygons/lines/points in a JSON format, making it easier to store in the database as a string and load it to the map when needed.

- **Pollution event characteristics**: This is a button in the map container where we can access the information about the pollution event, such as the volume of spilled oil, number of injured people, and type of HydroCarbon spilled.

- **Facilities**: The facilities are stored in the database. When the game starts, they are loaded into the client so that the facilities' markers are put in their respective location. A participant can interact with a facility to enter inside. While inside, he can access the email and draw/consult/edit the sensible and interdict areas. Also, every facility has a warehouse with equipment/consumables/vehicles, where the participant can load his inventory, get the right equipment to use for combat by loading it onto a vehicle and driving it to the combat location.

- **Routes**: Until this point, the player could move or drive to a place instantly. So, there was a need to give more realism to it by attributing a duration and distance based on the speed of the participant/vehicle. After some discussion, we also decided to generate an interface that simulates a GPS route and its details to give even more realism to it. It was hard to get the appropriate Service that could provide the information needed. After some time searching and trying very different approaches, we chose the MapBox API. This API receives the starting and destination points, the "traveler" whether it's a car or a person, and returns the coordinates of the route, the duration, and the distance. They cannot supply this type of information for boats, so for this type of vehicle, we manually calculate the distance and duration, the coordinates to reach the destination, and the route to travel is a straight line drawn to the map. This is a flaw to improve in future work because the boat's route can overlay land, like islands or other non-water areas.

- **Message/Email**: We also implemented the functionalities of sending/reading emails/messages. If a participant has a phone in his inventory, he can access it to read/send messages. The same for the emails, when he is inside a facility where he can access a computer, to do that. Both the email and phone interfaces are generated using templates even though their templates only serve to generate them. However, since they are on a template and not hidden in the main game interface, we managed to spare some resources and improve performance. With these functionalities, the participants are able to talk with each other in game.

- **Equipment Actions**: We started implementing the core actions that a participant can do with an equipment/consumable. These actions are stored in the database, and they have an equipment type, equipment State, name, and participant state associated. This way, we can load to the equipment's interface the appropriate actions depending on the context of the game. For a consumable, the only actions possible are to consume it to gain Health Points (HP), in the case it is in the participant's inventory, and to add to the participant's inventory in case it is in the facility warehouse. The actions implemented on an interaction with an equipment were:

  - **Vehicle's Inventory**: We implemented the functionality of having an

inventory on equipment that are vehicles. This inventory can contain other equipment that are not vehicles. This serves, for example, to move equipment from place A to B, using a vehicle to be faster. The inventory can be accessed from the equipment interface.

- **Carry/Release equipment**: We implemented the functionality of carrying equipment from place A to B. The participant can either choose to carry the equipment to a point chosen on the map or to a nearby vehicle. If he chooses to carry the equipment to a place on the map, the carrying is done on foot. This action is a group action, meaning that to do it, it needs more than one participant to help. When a player clicks carry, an interface appears where he can cancel by releasing the equipment or wait until the necessary people try to join to carry the equipment. When the necessary people to carry are ready, they can press the move button to carry the equipment to the aforementioned options, place, or vehicle. After choosing the option, an interface appears to simulate the time taken to do this action based on the duration and distance of the movement.
- **Fuel equipment**: We implemented the functionality to charge fuel on equipment that runs on diesel. To do this, the participant needs to have a fuel kit in its inventory. After pressing the fuel action, an interface appears to simulate the time taken to do this action. After this, the autonomy of the equipment increases.
- **Unload**: This action is the opposite of carrying equipment to vehicles. When accessing a piece of equipment on a vehicle's inventory, the player can choose to unload it to have the chosen equipment out of the respective vehicle to a place nearby.
- **Repair**: This functionality refers to repairing equipment. This can only happen if the participant has a repair kit with him. After pressing the repair action, an interface appears to simulate the time taken to do this action. After this, the health of the equipment increases.
- **Place/Remove boom**: With the functionalities mentioned above, we were able to do the following: carry a boom from a facility into a car; drive the car from the facility into the combat area; unload the boom to land; carry the boom to a boat; enter the boat and drive it to next to the spilled oil. With this, we implemented the action of placing a boom in the spilled oil to contain it. This is also a group action, so the process of doing the action is similar to the carry action. The participants choose to place the boom. When the necessary participants join in to place the boom, they can finish the action by drawing the boom on the map, similar to drawing sensible/interdict areas. After this, the boom appears on the map constantly. If a participant is near the placed boom, he can remove it as well if he has the necessary number of participants with him, as it is a group action as well. Both the placing and removing actions needed to be communicated with the Digital Twin component to report the action to MOHID, but at this moment, this wasn't possible as there were some problems on the Digital Twin Component.

This was the hardest working sprint as there were a lot of functionalities to implement. In the testing phase, we discovered some bugs in the functioning of some actions. We didn't manage to have a working version with enough functionalities to simulate an exercise to combat a pollution event.

6.2.4 Sprint 4

The final Sprint was dedicated to getting the final version of the project and deploying it on the provided server at DEI, executing the final tests, presenting it to the client, and a final meeting with the tutors. In this sprint, we implemented the final touches in the project, like some in-game actions that were still not working and others that did not exist, so we could have enough functionalities to simulate a combat exercise.

Because of these actions, the interface changed a little, and the core actions appear as buttons on the page, such as speaking, notifications, accessing the map, and resting. The speak action was implemented in this Sprint only. By pressing a button, the participant can write a message and send it. Everyone who is close by, will receive a notification that someone spoke. The resting action was not implemented as well until this point. Now, When the participant clicks the rest button, an interface of resting appears, and in the end, the user gains some HP. The notifications area was not implemented as well at this point. When a participant received a notification and closed it, he couldn't reaccess the notifications. So, we implemented a component where he has access to past notifications. There was a need to implement this since, with the speak notifications and the use of templates, there was a flaw that happened When one participant spoke and then another participant spoke as well. The notification would be replaced by the second participant who spoke, and the players could miss some important information. We finally could put a barrier since we fully implemented the communication between Digital Twin, MOHID, and the other components. After this, we started the deployment of the project on the provided server, which generated a lot of problems and took way more time than we thought we would need to do it. These problems were happening because of misconfigurations and uninstalled technologies in the server and small and almost undetectable errors where a letter being in uppercase would not give an error in localhost but in the server would generate one.

After this, we presented the final version to the client, where the feedback was really good and better than expected because some actions were still not properly working.

To finish the development phase, the final meeting with the tutors happened with the objective of them playing and experiencing the game so that they can give their feedback about the game and about each component individually for the dissertation's writing purposes. At the end of the development phase, we started writing the dissertation.

6.3 UI Generation- Pratical Examples

This section presents the results of the generation of interfaces and their details for the most important functionalities using Templates and the results of the proposed model.

6.3.1 Templates

As mentioned before, the Interface Generation was accomplished with the use of templates. Figure 6.2 represents the process of generating an equipment's interface using templates.

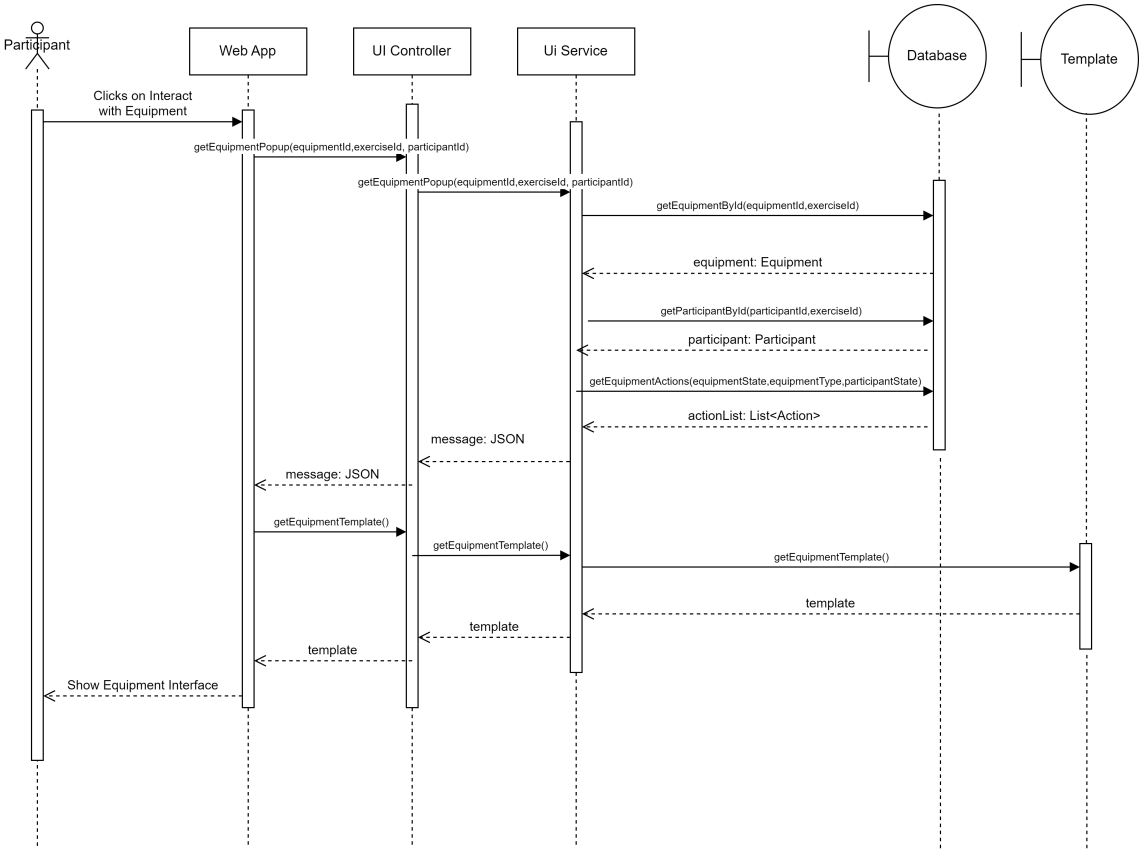


Figure 6.2: Generation of an equipment Interface using Templates

As we can see by analyzing Figure 6.2, there is an action in the client that triggers a request for an equipment interface to the server at *getEquipmentPopup*. This request contains the participant's Id, the equipment's Id, and the exercise's Id. The request is received by a Spring Boot Controller mapped at that endpoint. The Controller searches for the equipment in the database and retrieves its information. Then, it makes a request to the database to retrieve the possible actions to perform with the equipment based on its type, state, and the participant's state. The obtained information is converted to a JSON String. Then, an object is created with the Json String and the Label *EQUIPMENTPOPUP*. This object is sent to

the WebSocket Topic respective to the participant at */exerciseId/participantId*. The WebSocket Broker reads the message and executes the code where the message's Label equals *EQUIPMENTPOPUP*. Then, the code proceeds to call a function that fetches the endpoint *equipmentTemplate* to retrieve the appropriate template. In the server, there is a controller that maps this endpoint. It searches for the template and returns it. Templates are HTML files that contain divisions or variables appropriate for their goal; for example, if it is an interface for a piece of equipment, it already contains a division for placing the equipment's status, an inventory button, and a division where the actions available to be carried out on the equipment will be located. Then the client dynamically loads the data to these divisions. After loading it, presents the generated interface to the user.

The following Figures represent the main interfaces of the game, with detailed descriptions.

6.3.1.1 Main Components of the Interface

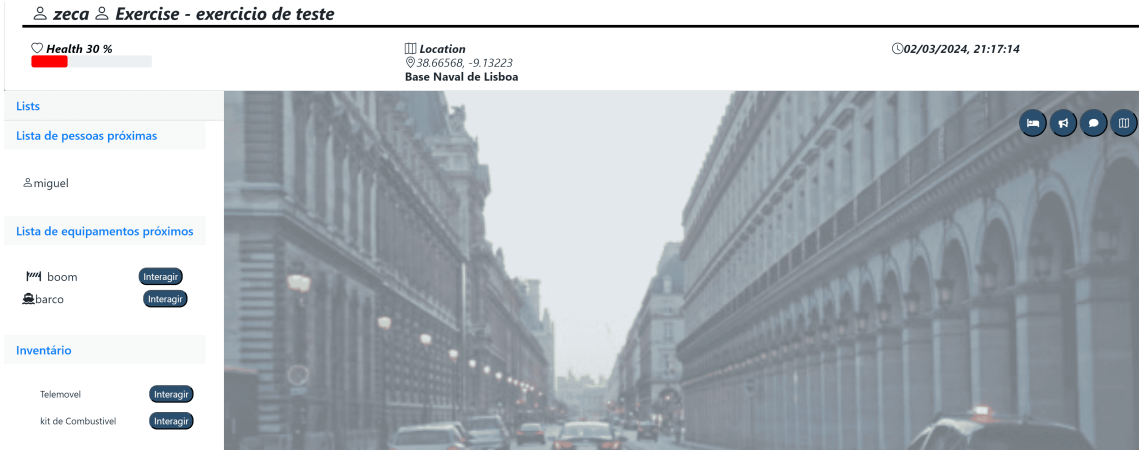


Figure 6.3: Main Components of the Interface

Figure 6.3 contains the main interfaces of the game. These components are fixed and always visible to the player.

On the left side of the screen, we can see: the list of nearby participants where the player can see who is nearby to help do a group task like, for example, carrying a piece of equipment or placing a boom; the list of nearby equipment where he has access to the nearby equipment where he can interact with and perform actions with it; and the inventory that contains equipment to use, for example, a cellphone, or consumables to use, for example, a fuel kit to fuel a vehicle, or food to gain HP.

In the top part of the screen is the participant's status which gives information and context about the game's current state. It contains his name, the name of the exercise being played, his health bar, his location, and the current date and time of the game in the simulation.

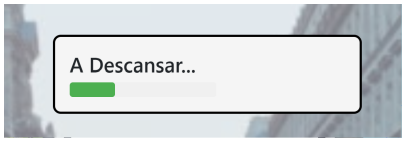
In the right part of the screen are the actions always available for the player to

perform. These include the rest action to gain HP, the speak action where he can write a message for every player nearby to receive, the notifications area where he receives the notifications of other participants who spoke, and the button that accesses the map.

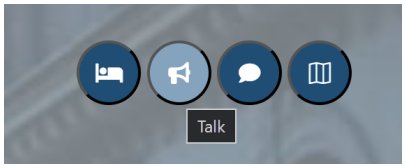
The background image also updates according to the place where the participant is. If he is inside a facility, the background image switches to an image that resembles an office. If he is inside a vehicle, the background image switches to an image that resembles the vehicle he is in, like a boat or a vehicle. The background image serves to help the player know where he is at and give more context about the game.

6.3.1.2 Available Actions

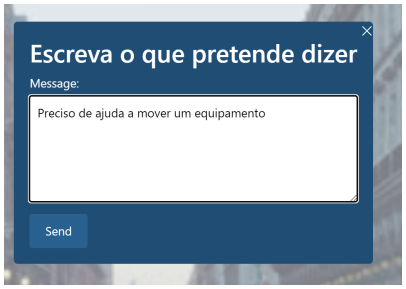
(a) Rest Action



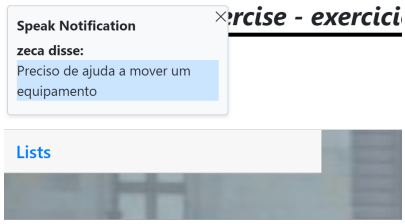
(b) Resting



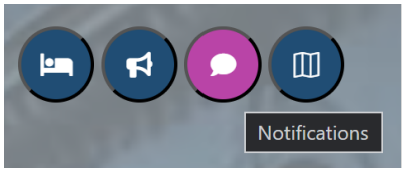
(c) Speak Action



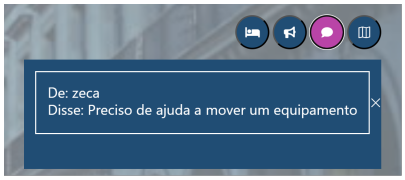
(d) Writing the speak message



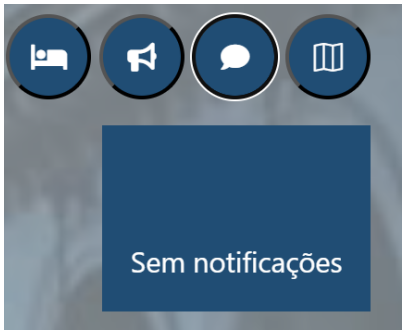
(e) Speak notification



(f) Speak Notifications



(g) Speak Notification

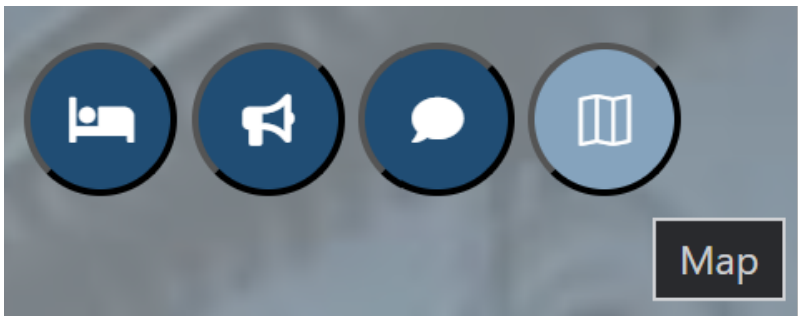


(h) Notifications empty

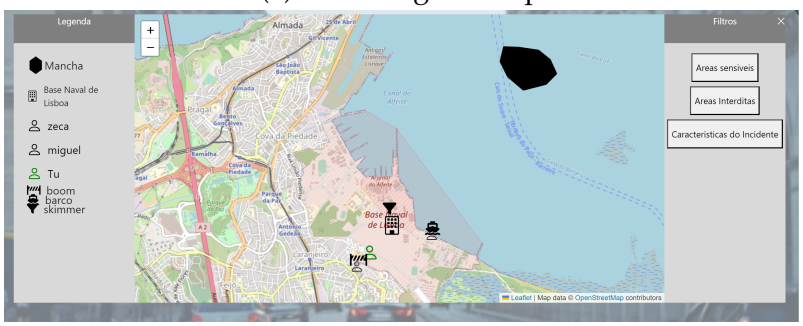
Figure 6.4: Available Actions interfaces

Figure 6.4 represents the actions that are always available for the participant to do. Figure 6.4a and Figure 6.4b show what happens when a participant clicks to rest. An interface to simulate the time needed to rest appears, and the participant gains HP. Figure 6.4c shows the speak action. When pressed, the user writes a message as shown in Figure 6.4d and the participants nearby receive a notification as presented in 6.4e. This notification can also be accessed in the notifications area. When the participant receives a speak message, the notification icon turns pink as shown in Figure 6.4f, and inside appears every speak notification as presented in 6.4g. The participant can choose to clear the notifications as shown in Figure 6.4h. The map button serves to access the map as shown below in Figure 6.5.

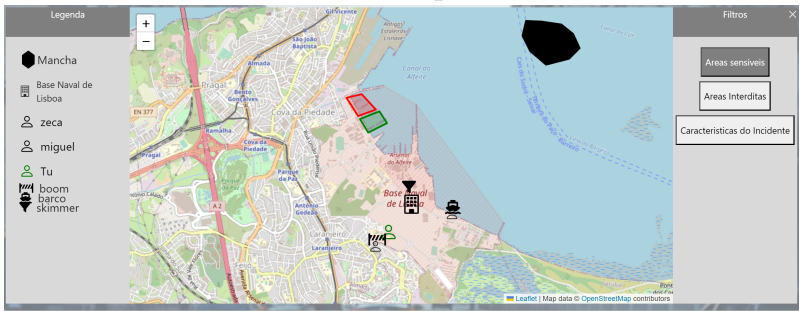
6.3.1.3 Map



(a) Accessing the map



(b) Map



(c) Sensible Areas in the map

(d) Characteristics of the incident

Figure 6.5: Map interfaces

Figure 6.5 contains the interfaces related to the map. Figure 6.5a shows where the player can access the map. It's in the button placed in the actions' area, as was shown before. Figure 6.5b shows the general aspect of the map container. On the left side of the container is the legend of the map to help the player understand better what is marked on the map. The central area contains the map

itself. Every marker placed on the map can be a participant, equipment, Facility, or the spilled oil. The participant can drag the map at will. Besides providing information about the current state of the game, the map also serves for the participants/equipment to move. A participant can move by double-clicking the map. If the participant is inside a vehicle, he can also double-click the map to drive somewhere. On the right side of the container it's the layers that the player can apply to the map. Figure 6.5c shows an example of the participant applying the Sensible Areas to the map. The interdict areas are shown the same as the Sensible Areas. Figure 6.5d shows the last layer that the player can access, which is the characteristics of the incident, where the participant can access the important information about the incident, like the number of injured people, the volume of Hydrocarbon spilled and the type of spilled oil.

6.3.1.4 Route Interface

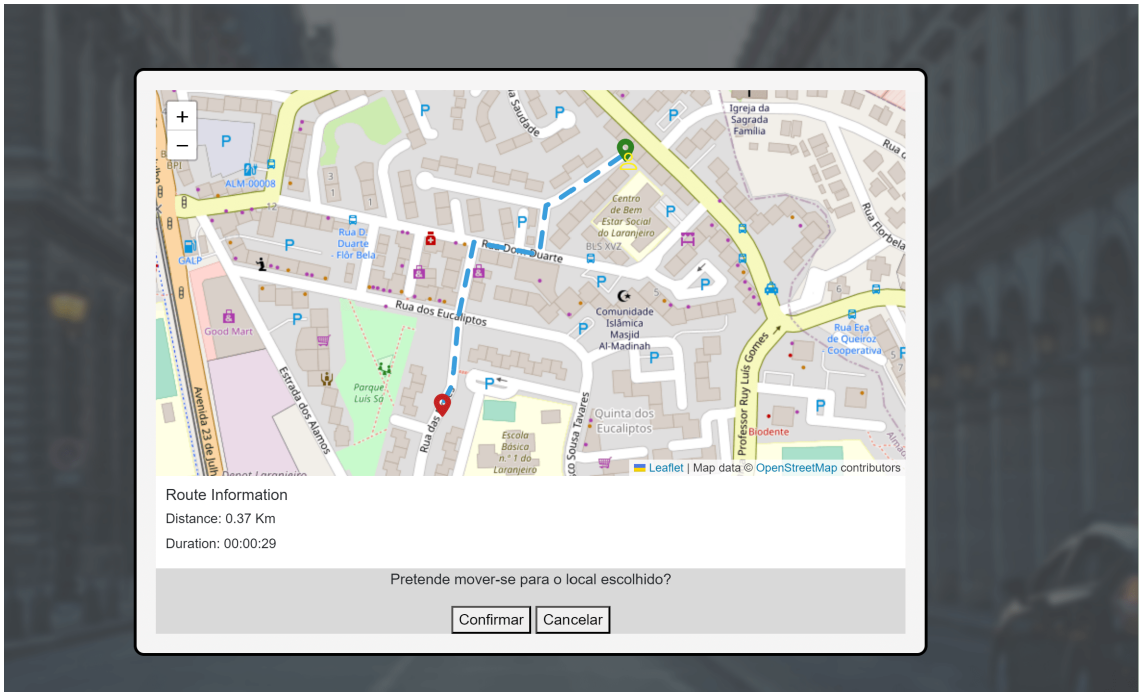
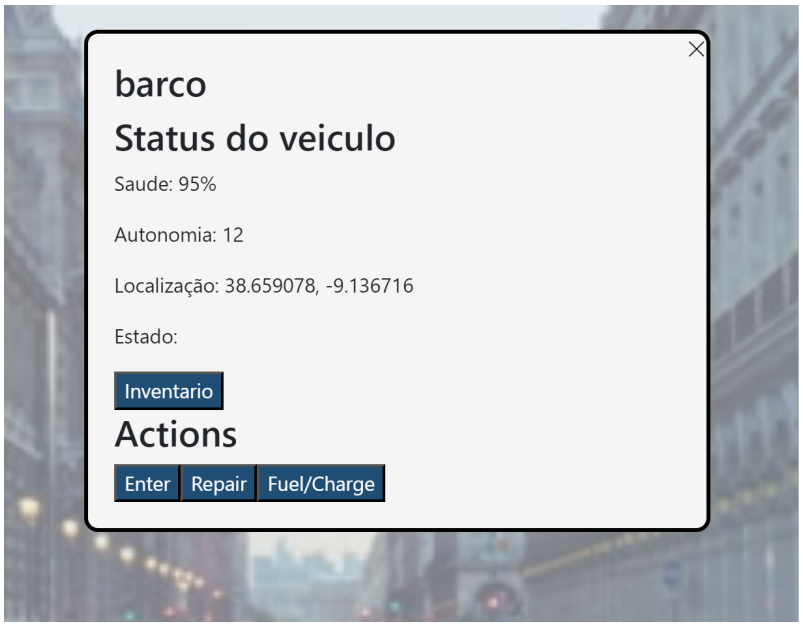


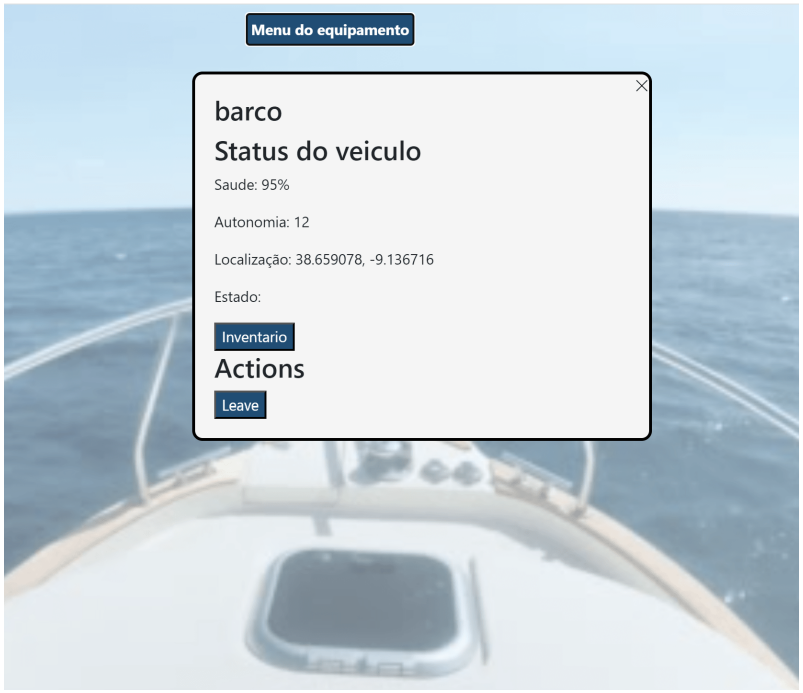
Figure 6.6: Interface with the Gps Route

Figure 6.6 shows the generated interface when a participant wants to move. When he double-clicks the map to move, the starting and destination points are sent to the server. As mentioned before, the server makes an API request to an online Service to get the GPS information. The server builds the necessary information, like the coordinates of the route, the distance, and the travel duration. The client receives this information and creates this interface. This interface, as shown in the Figure, contains the distance and duration of the travel and the route the participant will travel. The interface also contains two buttons where the participant can cancel or confirm the travel. If he confirms, the Buttons disappear, and this interface updates dynamically periodically with the new coordinates of the participant and the new distance and duration.

6.3.1.5 Equipment Interface



(a) Boat Interface



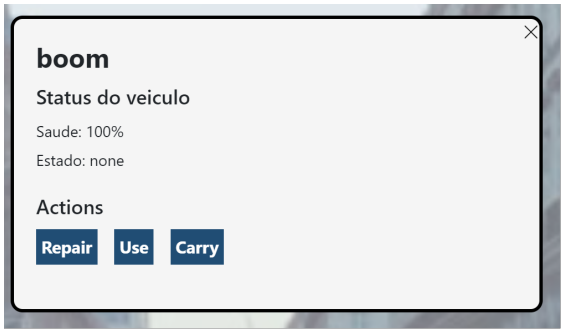
(b) Boat Interface while inside

Figure 6.7: Vehicle Interfaces

Figure 6.7 shows the interfaces representing equipment, in this case, a boat. As we can see in Figure 6.7a, the interface contains on the top the name of the equipment. Then, it follows the area containing the vehicle's status, like its health, autonomy, and location. Next follows the inventory button, where the player can access the inventory of the equipment. This inventory contains other equipment and consumables that the participant can interact with. At the bottom is an area

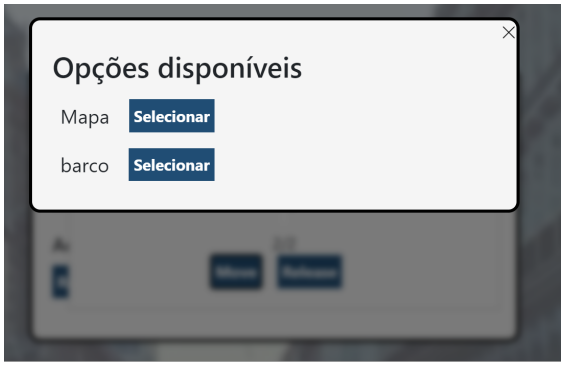
containing the actions that the participant can perform with the equipment. The actions are stored in the database and appear dynamically in the interface based on the equipment's state and type and the participant's state. Figure 6.7b shows how the interface is updated when the participant enters the boat. The actions change to Leave. Also, as mentioned before, the background image changed to a boat to give more context to the game.

6.3.1.6 Carry Equipment Action

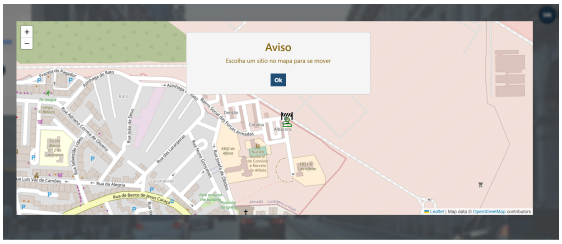


(a) Interacting with a boom

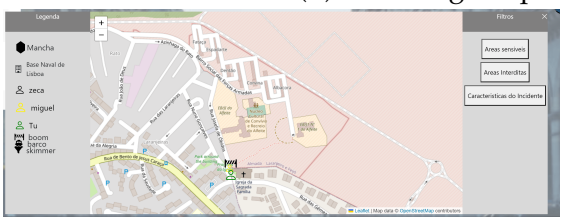
(b) Waiting for other participants to help carry the boom



(c) Available options to move



(d) Choosing the place in the map to move



(e)

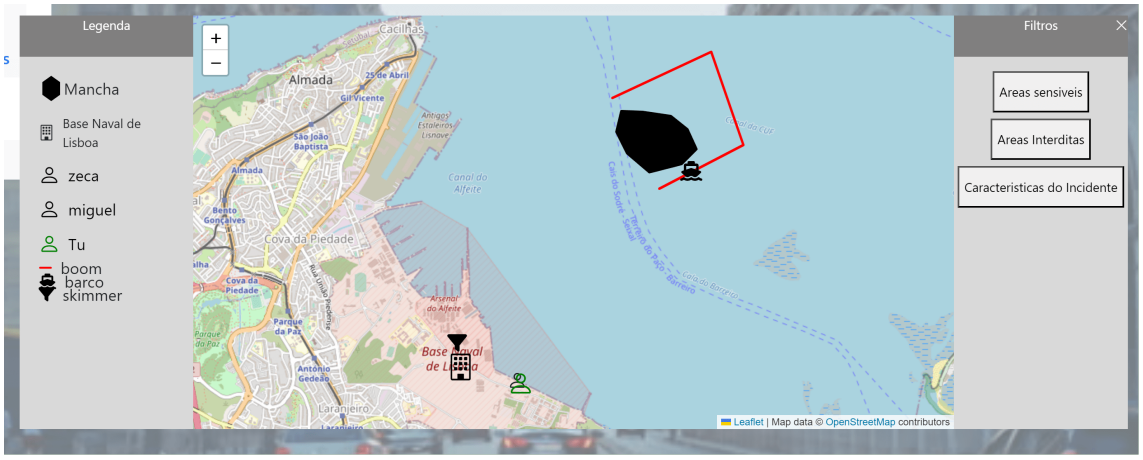
Figure 6.8: Map updated with new location of the boom and the participants that carried it

Figure 6.8 contains the interfaces related to carrying a boom. Figure 6.8a represents the interaction between a participant and a boom. He can perform the actions: Repair the equipment if he has a repair kit in his inventory so that the equipment gains HP; Use to use the equipment, in this case, placing the boom; Carry to carry the boom to either a point chosen in the map or a vehicle nearby.

When he presses to carry, an interface appears where he has to wait for the necessary participants to carry, as well as it is an action that requires more than one participant to do, as shown in 6.8b. Other players can join by pressing the carry action in the equipment's interface. After the necessary participants join in, the move button can be pressed to choose one of the available options as presented in Figure 6.8c. These options are generated at the moment of the request. The map is always an option. The vehicles depend whether there are any nearby or not. In this example, the participant wants to carry the boom using the map as shown in Figure 6.8d. He proceeds to choose a point on the map. Then, the map updates periodically depending on the duration and distance of the travel, as mentioned before. Figure 6.8e shows the map updated after the carrying action. The participants and the equipment being carried are moved to the chosen place.

6.3.1.7 Placing a Boom

(a) Placing a Boom



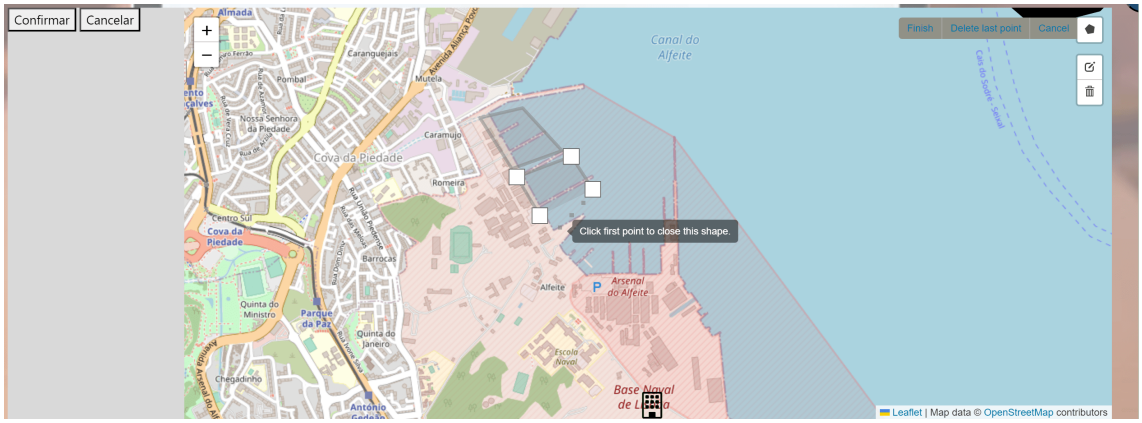
(b) The result on the map

Figure 6.9: Placing a barrier and its result on the Map

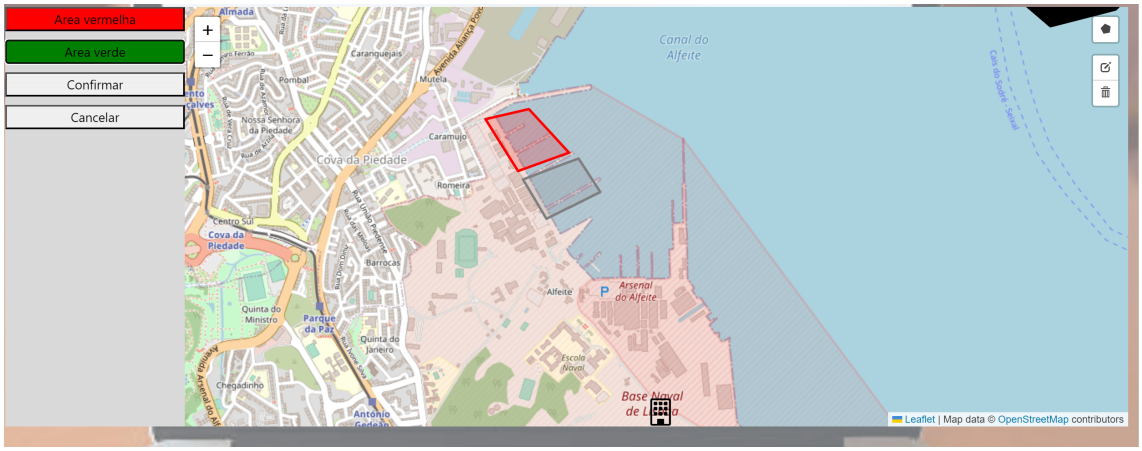
Figure 6.9 shows the process of placing a boom in the water and its effect on the map. When a participant is inside a boat containing a boom in its inventory,

the participant can interact with the boom to place it. As this action is a group task, an interface for waiting for other participants appears similar to Figure 6.8b. When the necessary participants are available, the participant can start drawing on the map the points where he wants the boom to be placed, as shown in Figure 6.9b. After doing this, the map updates for every participant in the game with the boom appearing on the map, as presented in Figure 6.9b.

6.3.1.8 Sensible/Interdict Areas



(a) Drawing an area



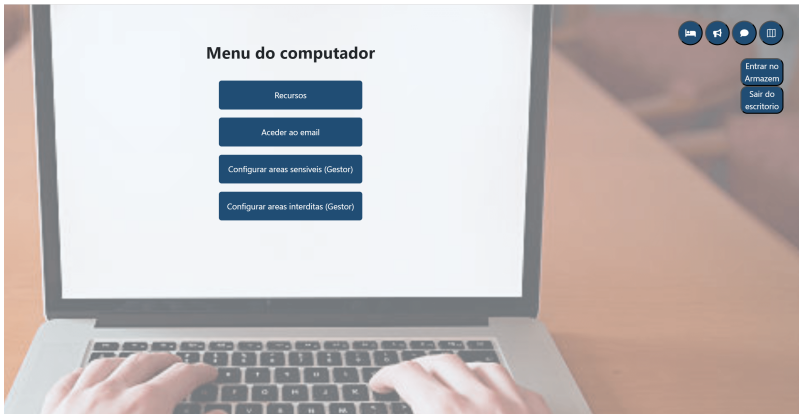
(b) Coloring the areas

Figure 6.10: Sensible/Interdict Areas

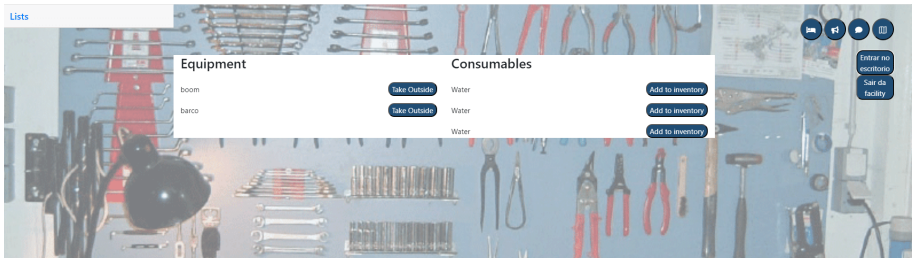
Figure 6.10 shows the process of drawing an area and its coloring. The process of drawing the areas is very similar to placing a boom. As shown in Figure 6.10a, the player chooses the points to form a polygon. Then, he has to color the areas to distinguish them, as presented in 6.10b. Then, the participants can access these areas by applying the respective filters as shown before in Figure 6.5c.

6.3.1.9 Facilities

(a) Interacting with a facility on the map



(b) Inside a facility

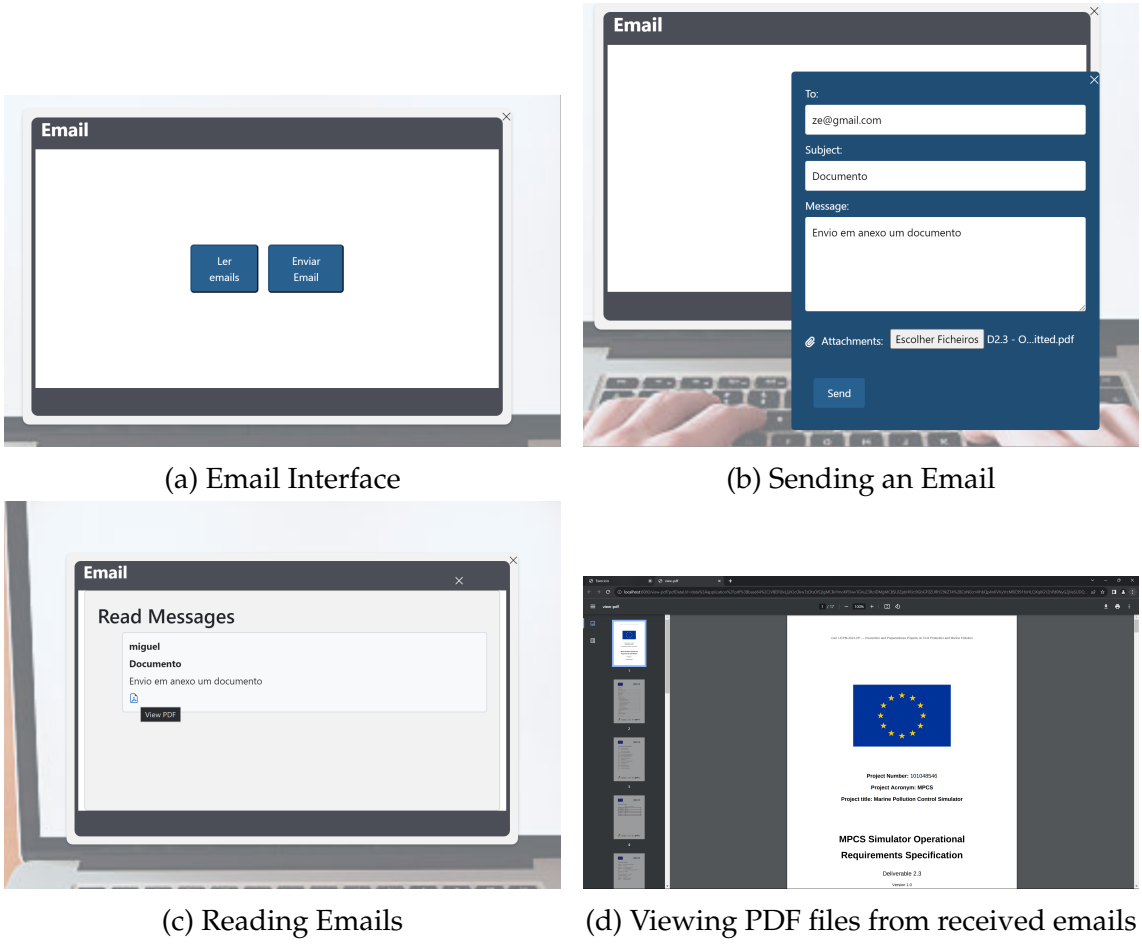


(c) Inside a facility's warehouse

Figure 6.11: Facilities Interfaces

Figure 6.11 shows interfaces related to facilities. On the map, if the participant is close enough, he can interact with an interface by clicking on the marker. Then, he can either enter the facility's warehouse or the facility itself. Figure 6.11b shows how the interface looks like inside a facility. The background changed to an office-lookalike image. Here, the participant can access the email, draw/create/edit the Sensible/interdict Areas, and access other resources that contain features that are not available at the moment. Two Buttons also appeared in the right part of the interface, where the participant can click to exit the Facility or enter the warehouse. Figure 6.11c presents the Facility's warehouse. Here, there is a table showing the equipment and consumables inside the Facility. The player can add consumables to his inventory or interact with equipment to take them outside to use them.

6.3.1.10 Email



(a) Email Interface

(b) Sending an Email

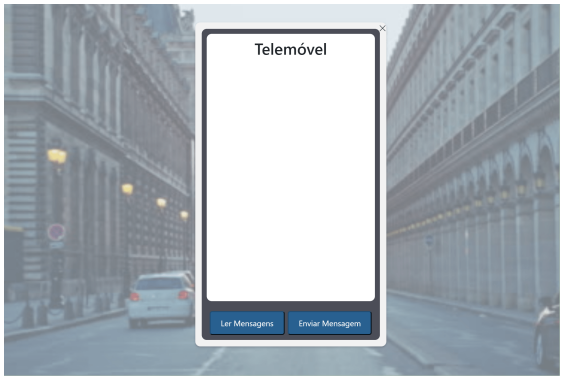
(c) Reading Emails

(d) Viewing PDF files from received emails

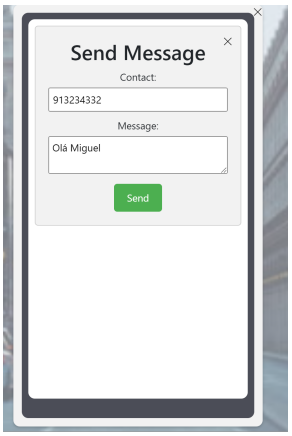
Figure 6.12: Email Interfaces

Figure 6.12 contains the interfaces related to the email. This interface appears when a participant is inside a Facility and clicks on the access email button that appears in Figure 6.11b. Then, he can: send an email by writing the email of the receiver, the subject, and the body of the email. He can also attach files; or he can choose to read emails as shown in Figure 6.12c. The received emails can contain pdf files that he can see by pressing the attached file, as shown in Figure 6.12d

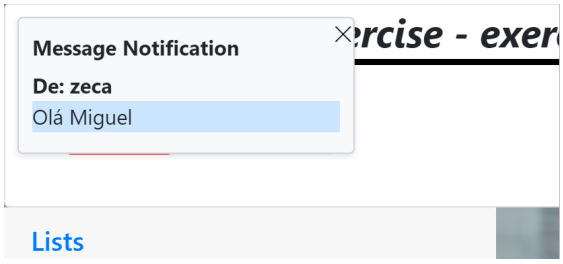
6.3.1.11 Messages



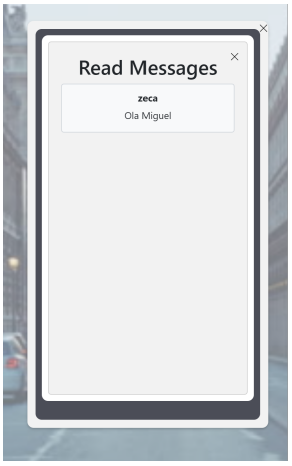
(a) Phone Interface



(b) Sending a Message



(c) Message Notification



(d) Reading Messages

Figure 6.13: Message Interfaces

Figure 6.13 contains the interfaces related to the Messages. Every participant has a phone in his inventory. Besides emailing, this is how to communicate with other participants who are not nearby. When interacting with the phone in the inventory, the phone interface appears as shown in Figure 6.13a. The participant can either choose to send a message and proceed to choose the receiver(s) and write the body of the text message as it is shown in Figure 6.13b or read the messages received, as shown in 6.13c. Figure 6.13c shows how the player is notified of a new message. Every time he receives a message, a notification appears indicating the sender's name and the content of the message.

6.3.2 Automatic UI Generation Proposal

This subsection presents the development process of the automatic user interface generation model proposed in 5.3. Instead of only using templates with empty components and filling them when requested, this model takes a JSON configuration file, where the game's administrator can configure how the interface is going to be like, such as the fields that it is going to contain, as well as their appearance. This way, there are a lot of different interfaces that can be generated without the administrators having to touch the code. This model was only applied for an equipment interface since they are the ones that can be more diverse. Figure 6.14 presents a sequential diagram showing how the generation process works for an equipment interface.

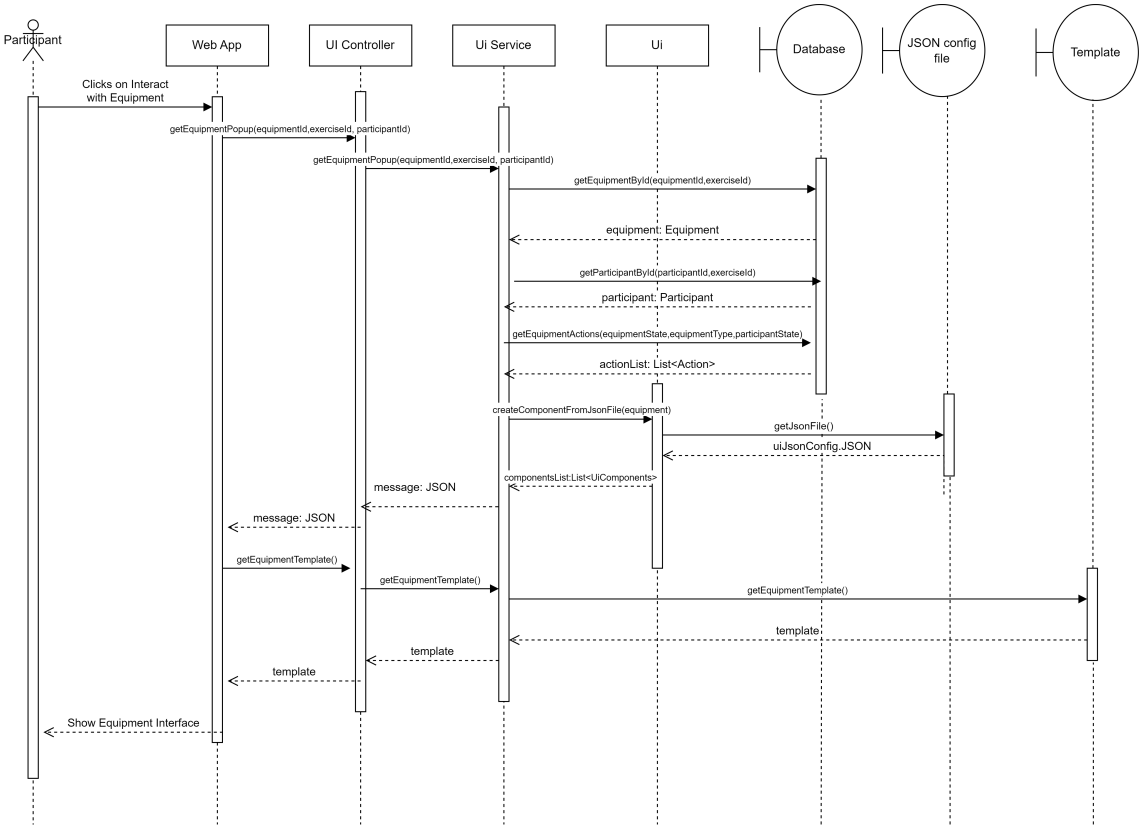


Figure 6.14: Sequential diagram of the automatic user interface model

By analyzing Figure 6.14, when the user clicks to interact with an equipment, the web app calls the function *getEquipmentPopup* that makes a request to an endpoint, passing as arguments the equipment id, the exercise id and the participant id, to get the information about the equipment. The spring boot controller mapped at that endpoint, calls a method to get that information using the same arguments received. This method makes three requests to the database, one for the equipment, another for the participant that is interacting, and another to get the possible actions to perform with the equipment, based on its type and state, and the participant's state. After getting all this information, calls a method from the class *UI* and passes the equipment as an argument. This method reads the JSON file, and compares all the fields that the equipment has with the ones de-

scribed in the JSON file. From the fields that match, it creates a list of the components. These components are none other than the description for each field in the JSON file, such as its data type and the set of properties, as we can see, for example, in Figure 6.15. This list essentially contains a list of JSON objects, each element describing one field of the equipment. A message is then created with the label *GETEQUIPMENTPOPUP*, the list of components, and the list of actions, and returned to the spring boot controller to return it to the web app. When the web app receives this message, makes another request for the equipment template. This template is empty, only containing a division for the action buttons to be. The spring boot controller receives the request and calls a method to search for and return the template. When the web app receives the template, calls a function to analyze the list of components received prior and starts building the HTML components based on the JSON description of each one. This process includes analyzing the data type and the set of properties of each field. Then, since the template is empty, it just adds the HTML component to the template. After finishing processing every component, shows the interface to the user.

Figure 6.15 represents a sample of the JSON configuration file for a maritime vehicle. Since this file is big, Figure 6.15 presents an example of the field name that has a text data type, and Figure 6.16 presents an example of the field health that has a range data type. These examples serve to show the small differences that there are between a text field and range fields in their configuration.

Figure 6.15: First sample of the JSON configuration file

As we can see by analyzing Figure 6.15, the JSON file is divided by equipment types. Then, each equipment type has an array of components that will be represented in the interface to be generated. Each component contains the details about the field's name, the data type, and the set of properties that will define the visual aspect of the respective component to be generated, as explained before. This example shows the component's configuration for the field name.

```
"name": "health",
"type": "Range",
"values": [
  {
    "name": "MinValue",
    "value": 0
  },
  {
    "name": "MaxValue",
    "value": 100
  },
  {
    "name": "Units",
    "value": "%"
  },
  {
    "name": "DisplayType",
    "value": "Bar"
  }
],
"properties": [
  {
    "name": "Priority",
    "value": "1"
  },
  {
    "name": "Interaction",
    "value": "none"
  },
  {
    "name": "Customization",
    "value": "none"
  },
  {
    "name": "Group",
    "value": "1"
  },
  {
    "name": "Relevance",
    "value": "High"
  }
```


Figure 6.16: Second sample of the JSON configuration file

By analyzing Figure 6.16 we can see the configuration for the autonomy field. The difference between the range fields and the text fields is that we have to specify the minimum and maximum values of the range, as well as the units. When implementing this model, a new property for the range fields, the DisplayType, was also added. This property has three options and allows the administrator to choose a special interface for a field. It can have the values: *Gauge* for representing a field as a gauge component; *Bar* to represent the field as a bar, or *none* if the administrator intends to let the interface be generated completely automatically.

With the description in this file plus the methods that interpret the JSON description mentioned before 6.14, we can generate interfaces that can look like the

examples presented in Figures 6.17, 6.18, 6.19.

Table 6.1 presents the input data for the interface represented in Figure 6.17.

Name	Type	Priority	Interaction	Customization	Group	Relevance
name	Text	1	none	none	1	Medium
brand	Text	2	none	none	1	Medium
health	Range	4	none	none	none	Low
autonomy	Range	5	none	none	none	High
speed	Range	3	select	none	2	High

Table 6.1: Input data for Figure 6.17

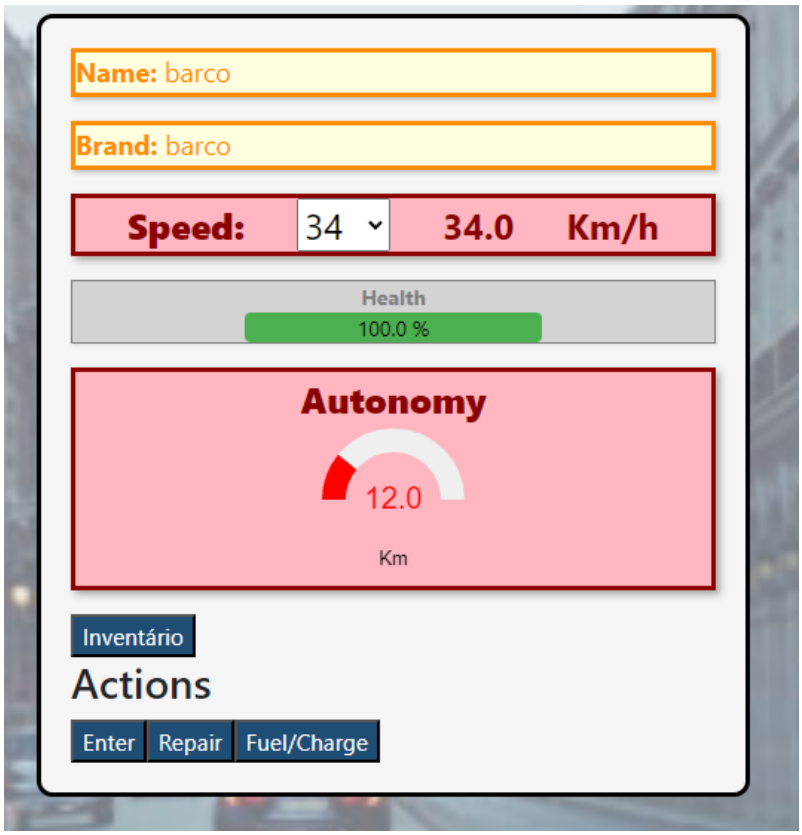


Figure 6.17: First example of the generated interface

We can see how the set of properties generated the interface by analyzing Table 6.1 and Figure 6.17. The components appear in the order defined by the priority property. Then, only the speed had an interaction value of *select*, resulting in a select component to choose the most adequate value. In this example, there was no customization in any field. Then there are two groups: Group 1 which joins name and brand together, and group 2, which only has speed. In this example, it's not very clear the usage of groups because it just looks like they are sorted by priority. We can see better the usage of groups in Figure 6.18 and Table 6.2. Finally, autonomy and speed are the most noticeable components in the interface since they have a relevance value of *High*, followed by name and brand, which have a relevance value of *medium*. Speed is less noticeable since it has a relevance

value of *Low*. For this interface, the autonomy also had a DisplayType value of *Gauge* and health has a DisplayType of *Bar*.

Table 6.2 presents the input data for the interface represented in Figure 6.18.

Name	Type	Priority	Interaction	Customization	Group	Relevance
name	Text	4	none	red, impact	none	Low
brand	Text	5	none	none	1	High
health	Range	4	none	none	3	High
autonomy	Range	7	none	none	2	Medium
speed	Range	2	drag	none	2	Low

Table 6.2: Input data for Figure 6.18

Figure 6.18: Second example of the generated interface

As we can see by analyzing Table 6.2 and Figure 6.18, we can immediately notice that although the autonomy field has a priority of 7, it appears as the second element of the interface. This happens because when groups are involved, the grouping property prioritizes sorting the elements. This means that the component of a group with the highest priority creates the group. Then, the other components of the group are aggregated, in a sorted way, next to the component that created the group, ignoring the priority of components of other groups. this makes it possible for the components of a group to be together regardless of the priority of other components. Then we can see that the speed has an interaction

value of *drag*, resulting in a slider component. Then, we can see that the component brand has a customization value of *red, impact*, resulting in altering its font and color to the specified. At last, once again, the relevance property made the components with higher relevance more noticeable than the others.

Table 6.3 presents the input data for the interface represented in Figure 6.19.

Name	Type	Priority	Interaction	Customization	Group	Relevance
name	Text	2	none	none	2	High
brand	Text	4	none	red, impact	3	Medium
health	Range	1	none	none	1	High
autonomy	Range	1	none	none	2	Low
speed	Range	2	select	none	1	Medium

Table 6.3: Input data for Figure 6.19

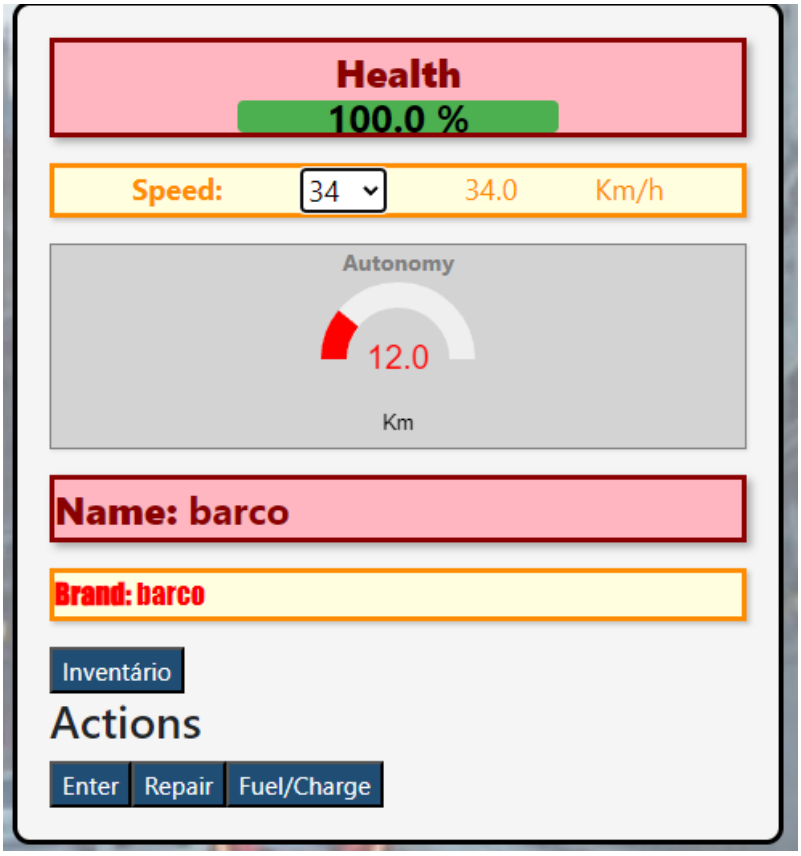


Figure 6.19: Third example of the generated interface

As we can see by analyzing Table 6.3 and Figure 6.19, we can see the group property being applied again together with the priority property. Although autonomy has priority 1, since the speed belonged to group 1, it appeared first so that it could be joined with its group. Then we can see again the interaction, customization, and relevance properties applied, as shown in the prior examples.

To summarize, this model can solve the problems encountered with the use of templates, such as enabling the administrator to customize the interface without

having to touch the code and also being able to generate any interface for any entity as long as it is configured in the JSON file, providing more flexibility and abstraction to the generation of interfaces. Since the interfaces will have a different visual aspect, depending on their configuration, the future simulator users will also have a clear notion of what equipment they will be interacting with since the customization serves to adequate the interface to its equipment, making the gameplay more intuitive and user friendly.

This model can be improved by adding other properties to define and generate even more distinct interfaces and adding more value options to the DisplayType property so that the administrator can have more contact with the generated interfaces, thus having even more customization. Another relevant improvement would be to create an interface in the game where the administrator could make all these configurations. Although JSON is an easy-readability format, it's more user-friendly to have a configuration interface where it's possible to make all these changes. Also, using this configuration interface would make it possible to change the visual aspect of the interfaces in-game without compiling and running the game every time there was an alteration to the JSON configuration file, making the model even more user-friendly.

Chapter 7

Evaluation

This chapter presents the details of the evaluations made to the MPCS project, the results obtained, and the result's discussion. To evaluate all the progress made, two evaluation studies were carried out. The first focused more on the usability, playability, and coordination of the MPCS project, serving as evaluation material for the MPCS project. The second was more focused on evaluating the game's interfaces, serving as evaluation material for this dissertation's project.

7.1 First Evaluation

The first evaluation served to evaluate the overall performance of the game. We wanted to test if the study participants could play the game and have a clear notion of the game's state, the surrounding environment, and the actions they could perform. The goal was to check if the gameplay was clear and intuitive, and if it was possible for the members of the study to coordinate between them when carrying out assigned tasks.

7.1.1 Study material

The first evaluation consisted of trying the MPCS's implemented functionalities with simultaneous users playing the game in the provided server. The study participants were the tutors who accompanied us while developing the MPCS project. A script was made for every study participant to follow in the game. Each script contained data about the in-game participant, such as the name, phone number, email, and role, and the assigned tasks to complete in the game. During the test, the development team members noted the time taken to execute each task from the script and the errors encountered.

At the end of the evaluation, the participants in the study were asked to fill in a Google form, where each member of each dissertation prepared some questions relating to their dissertation project and some questions relating to the project as a whole.

7.1.2 Results

With this evaluation, we found several errors in the MPCS project, such as failures in some functionalities that can ruin the gaming experience. We discovered that the server contained some errors that made every action on the system too slow. In addition, there were some errors in the communication between the components and the MOHID, making it difficult to provide enough realism to the game. However, by correcting these errors, the MPCS project is close to having the first version.

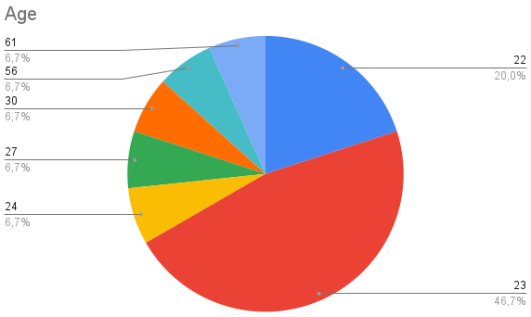
Concerning this dissertation project, some questions were asked about the usability of the interfaces, where the results were generally satisfactory, with room for improvement in certain aspects, such as some actions could be more easily accessible, better communication of the status, for example, the office where we are should appear in the profile, improvements in proportion to make better use of the screen area, graphic consistency (too many fonts and sizes), some imperceptible map representations, some dysfunctional overlays, some buttons too small, map plan does not take advantage of the useful area available, various elements of the exercise status could be visible so that the participant feels more updated about the game's state.

7.2 Second Evaluation

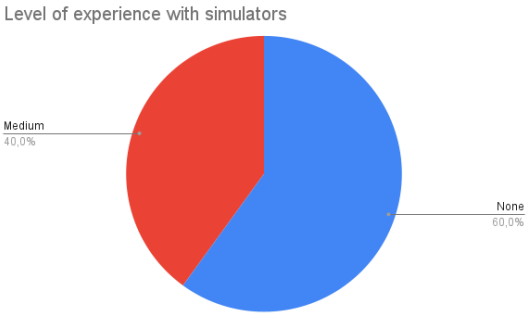
The second evaluation served to evaluate the game's interfaces. This evaluation aimed at testing the overall performance of this dissertation's project. The goal was to test the interfaces' usability, learnability, and effectiveness.

7.2.1 Study material

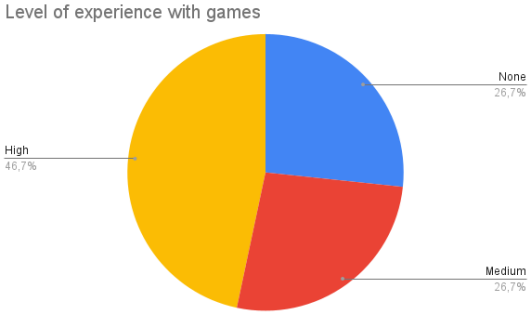
The ideal environment to conduct the study was for the people being tested to play the game by joining the server and accessing the game. However, due to some problems in the game's version deployed on the server, it wasn't possible to conduct the evaluation this way, not even in person. Therefore, the study was conducted via localhost, where I shared my screen, and the study participants guided me to do what they wanted. A series of tasks was created for them to execute, and for each task, the time taken, the tasks's completeness, and errors made when doing the task were noted. The errors counted were not system errors but errors in accomplishing the task, such as clicking on things that didn't need to be clicked on to fulfill the task and leaving the path that led to the task's success. Each test took about 20 minutes. Before each test, some questions were asked to frame the participant in a profile.



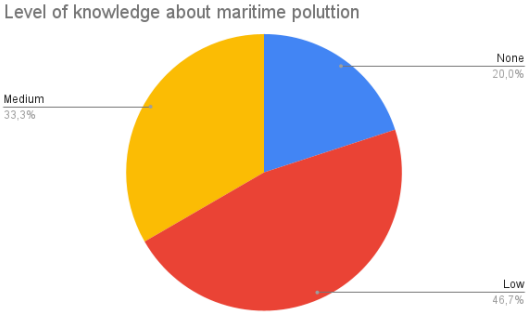
(a) Participants' age graph



(b) Participants' experience with simulators



(c) Participants' experience with games



(d) Participant's knowledge about maritime pollution events

Figure 7.1: Participants profiling graphs

The ideal profile should be future possible users of the simulator, but it was not possible to do it. By analyzing Figures 7.1a, 7.1b, 7.1c, and 7.1d, we can conclude that the profile of the study participants was similar. The age is around 23 years old. They either have none or medium experience with simulators. The experience with games varied a bit. Finally, in general, they all had little knowledge about maritime pollution events.

Before asking to do the tasks, it was presented to each study participant some context about the MPCS project and the marine pollution combat. Then, it was asked for the participants to do a brief analysis of the fixed interfaces shown in Figure 6.3 and also the map presented in 6.5, to see if they understood what was the meaning of each interface, and what they could possibly do with them. In general, they understood every interface besides the available actions on the right part of the screen, where there were some doubts about the icons chosen to represent the speak and notifications actions. After the analysis, we proceeded to start the execution of the tasks.

The tasks asked to perform were some of the core actions of the simulator that were completely functional so that the evaluation wasn't ruined. The following tasks were asked for the study participant to do:

- "The objective is to consult who are the people that are next to you and send a message to another user"
- "The objective is for you to enter a facility, and draw a sensible area"

- "The objective is for you to enter the facility's warehouse, add a consumable to the inventory and consume it and exit the facility"
- "The objective is to interact with the boom that is next to you and carry it into a vehicle, and then confirm that is in the vehicle's inventory"
- "The objective is to enter the vehicle that is next to you and drive it to near the coast"
- "The objective is to unload the boom from the vehicle to land "
- "The objective is to rest, and then speak to the participants that are next to you"

Once again, in each task, it was noted the time taken to fulfill the task, the completeness of the task, and the errors committed when executing the task. At the end of each task, the participants were asked if they had any doubts about the actions they were doing to know if they knew what was happening in the game, and, when necessary, more context was given to inform them about what is possible to do and also to compare with real-life scenarios to be more clear.

At the end of the execution of the tasks, the participants were asked to answer a Google form that contained some questions about the evaluation made, so that we could obtain information about their experience in the game.

7.2.2 Results

The Google forms that the study participants answered consisted of five questions using a Likert Scale, five questions using a Semantical Scale, and five open-answer questions.

Task	**Task's description**
Task 1	"The objective is to consult who are the people that are next to you and send a message to another user"
Task 2	"The objective is for you to enter a facility, and draw a sensible area"
Task 3	"The objective is for you to enter the facility's warehouse, add a consumable to the inventory and consume it and exit the facility"
Task 4	"The objective is to interact with the boom that is next to you and carry it into a vehicle, and then confirm that is in the vehicle's inventory"
Task 5	"The objective is to enter the vehicle that is next to you and drive it to near the coast"
Task 6	"The objective is to unload the boom from the vehicle to land "
Task 7	"The objective is to rest, and then speak to the participants that are next to you"

Table 7.1: Tasks's legend for the evaluation notes

Table 7.1 represents the legend of the tasks performed in the evaluation used in the annotation of the duration, errors, and completeness.

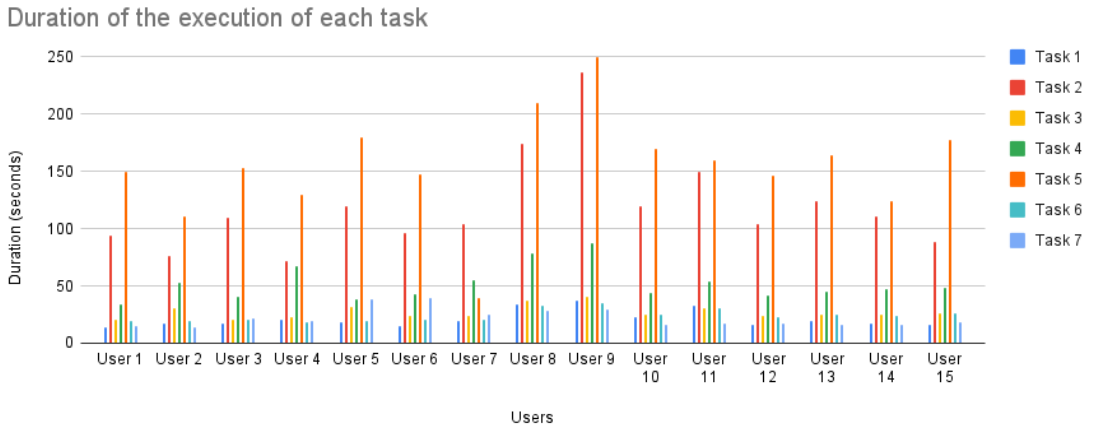


Figure 7.2: Duration taken in each task during the test

As we can see by analyzing Figure 7.2, the average duration of each task takes less than 30 seconds which is good since the tasks had a few steps do fulfill them. It's very clear that tasks number 2 and 5 took a lot of time for the participants to do, as it was expected since they were the most complex tasks of them all. However, these durations should be a lot less.

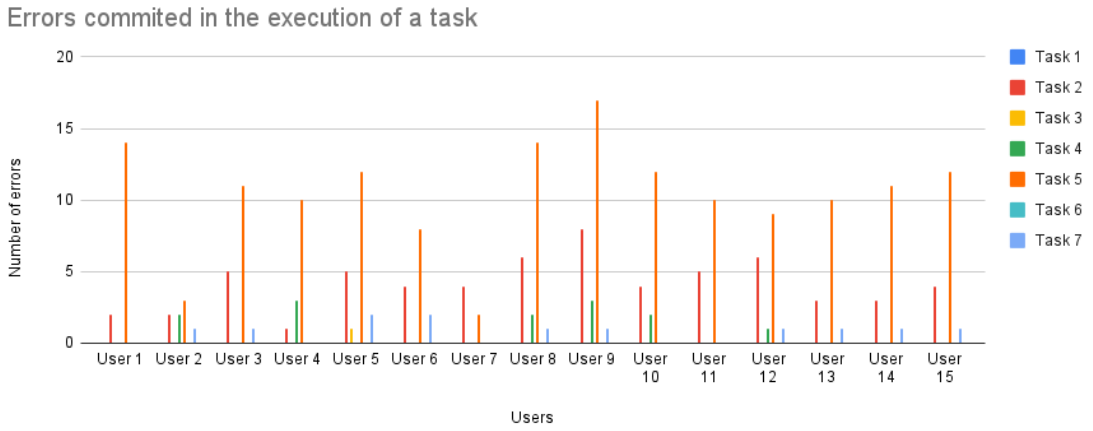
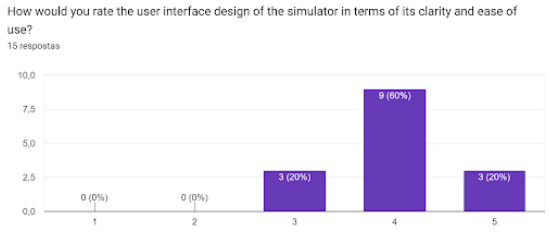


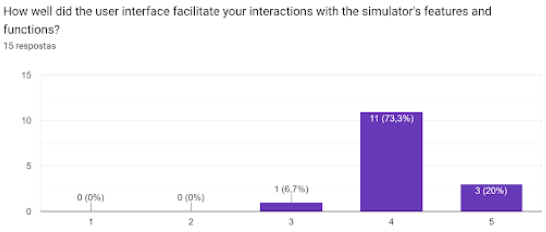
Figure 7.3: Errors counted in the execution of each task

As we can see by analyzing Figure 7.3, the number of errors is really small, which indicates that the participants in general understood quite well the interfaces. However, tasks 2 and 5 managed to generate the most errors among the participants. Also, besides resting and sleeping being a simple task, there were participants who didn't understand which icon represented the speak button, ending up pressing the notifications button, resulting in an error.

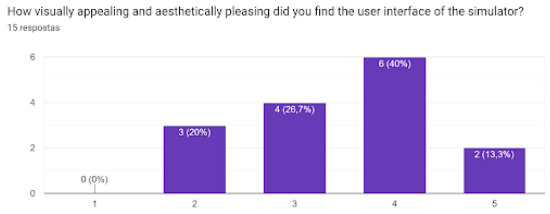
All the tasks were completed, so there was no need to create a graph with the completeness results.



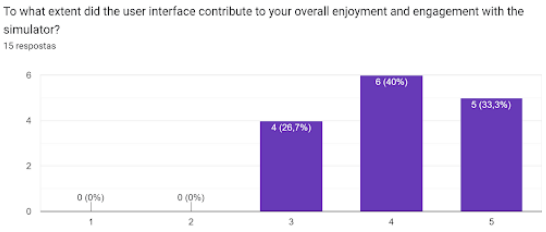
(a) First Likert Scale Question



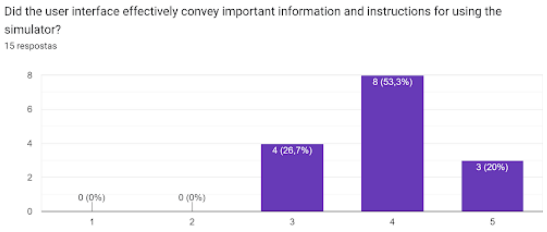
(b) Second Likert Scale Question



(c) Third Likert Scale Question



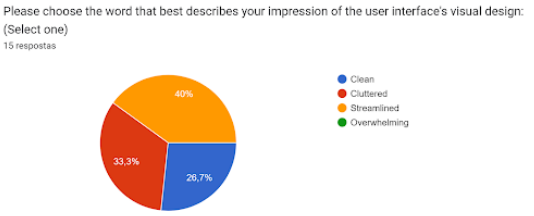
(d) Fourth Likert Scale Question



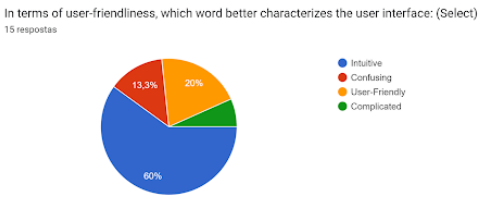
(e) Fifth Likert Scale Question

Figure 7.4: Likert Scale Questions

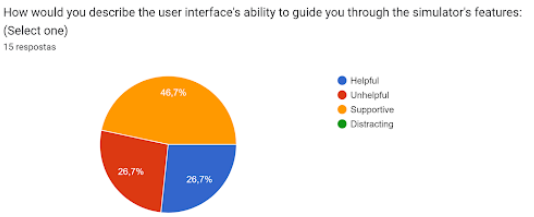
By analyzing the results presented in Figure 7.4, we can conclude that the participants' experience was not great, but it wasn't bad as well. In general, they found the interfaces to be clear and user-friendly. In terms of the visual aspect of the interfaces, the results were not good since most of the participants voted 3 or less. By analyzing these results we can conclude that the results were medium.



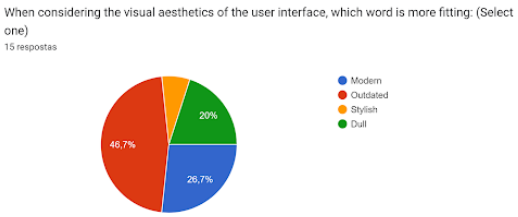
(a) First Semantical Scale Question



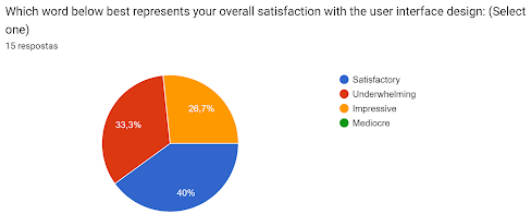
(b) Second Semantical Scale Question



(c) Third Semantical Scale Question



(d) Fourth Semantical Scale Question



(e) Fifth Semantical Scale Question

Figure 7.5: Semantical Scale Questions

By analyzing the results presented in Figure 7.5 most people found the interfaces to be simple and intuitive which is good, but nothing too impressive, and once again the visual aspect of the interfaces was not the best. However there was a diversity of opinions among the participants, making these results medium as well.

The open-answer questions were the following:

- "What specific aspects of the user interface design contributed to or detracted from your overall experience with the simulator?"
- "Were there any challenges or difficulties you encountered while navigating and using the user interface?"
- "How do you think the user interface could be further improved to enhance the overall usability and user experience?"
- "Did the user interface effectively support your understanding of the simulator's functionalities and objectives? Please explain."
- "Please provide any additional comments or suggestions related to the user interface design that you believe would be valuable for improving the simulator's usability and user experience."

The answers to the first question were good. The participants in general didn't have any serious complaints to make. They complimented the main interfaces which they found intuitive and supplied enough information. The most common complaints were about the font, color, and organization of the interfaces. Also, the icons that represented the available actions received some complaints about not being the most suitable.

The answers to the second question were also good. The only problem the participants complained about was when drawing the sensible area and when driving. They didn't find the methods to execute these actions the most suitable ones.

The answers to the third question provided some relevant information to update the game's interfaces. The advice consisted of having the interfaces in only one language which is an important failure of the interfaces, that was not noticed. The participants mentioned, once again, the need to update the visual aspect of the interfaces to a more appealing one, to improve the experience. They mentioned once again the better assignment of icons to the corresponding actions. They mentioned that there should be popups with help information so that the usability can be further improved. Finally, they continued to criticize the lack of consistency between font size and component size throughout the game. There was interesting advice about keeping the close lists of participants and equipment fixed, instead of closeable, and the inventory, instead of being a list, would be a button, that opened a pop-up that shows icons referring to equipment/consumables that the in-game participant brings with him.

The answers to the fourth question were really good. Besides the critics made about the overall experience, the participants were able to learn easily how the game works. They mentioned that the first touch with the game was a bit overwhelming, however, once they got adapted, the gameplay was smooth and intuitive.

The answers to the fifth question consisted generally of arranging the visual aspect of the interfaces into a more modern and appealing one. Then, they suggested changing the action to move from a double-click on the map to a drag or only one click since it makes more sense.

To summarize, this evaluation was critical, providing vital information about a first contact with the game. With this evaluation, it was possible to notice problems the development team members did not notice because we had already adapted to the game. In general, the participants did well in experiencing the game despite not having experience in this environment. They managed to fulfill almost every task in an acceptable duration without committing that many errors per task. Only the tasks of drawing a sensible area generated some confusion since they hadn't any notion of what a sensible area could be, which generated some hesitation in drawing them. The action to move, had big durations throughout every participant because they took some time to find that it was a double click in the map to move. After knowing, they admitted that it made sense, but without prior knowledge, they found it hard to find the trigger. The results were satisfactory in general since they found almost every interface user-friendly and intuitive So we can conclude that, in general, the functionalities and the corre-

sponding interfaces were well-designed and implemented, with still room to improve some visual aspects of the interfaces.

The suggestions from the study participants are well-taken and important, so they will probably be part of a future game update

Chapter 8

Conclusion

To conclude, having a wide variety of User Interface (UI)s in the MPCS creates a severe problem: the difficulty of manually developing the UIs that represent every interaction with the user, generating a considerable development effort.

To solve this problem, there was a need to research Automatic User Interface Generation Methods (AUIGM), to verify if user interfaces can be automatically created, significantly reducing the development effort.

The objectives for this dissertation's project were accomplished since we managed to implement a model of generating the interfaces in an abstract way using templates, which spared a lot of development time. While developing the game we came across the problem of lack of customization of the interfaces to be generated. So we developed and proposed a model capable of automatically generating the interfaces and also solving the problems of the model implemented in the MPCS project.

After conducting the user studies, we collected important information about the problems of the MPCS project and this dissertation's project. The evaluation was very important since we could have a first glance at the user experience, so we could find flaws in the overall game. After analyzing the results we concluded that this dissertation's project was a success with room to improve a few aspects, so that we can better meet the needs of the client and make the game experience more enjoyable.

For future work, there is still work to do so we can get a first version of the MPCS project, such as adding some functionalities not implemented yet, as well as fixing existing errors. In relation to this dissertation's project, the proposed model of generating interfaces using templates, has some improvements to make, such as the visual aspect of the interfaces. The model of automatically generating interfaces using a JSON configuration file can improve a lot by adding new properties and new options for customization to make the experience better. Also, an interface to configure the interfaces to be generated must be created to make the experience even more user-friendly. Finally, this model could be proposed to the client to be implemented to generate every interface in the game and therefore extend its capabilities.

References

[1] Sampaio Rui, Carrasqueira Manuel, and Daniel José. Marine pollution control simulator- functional requirements, 9 2022.

[2] Hallvard Trætteberg. Model-based user interface design, 5 2002.

[3] Krzysztof Gajos. Models in model-based user interface design. 3 2005.

[4] Nathalie Souchon and Jean Vanderdonckt. A review of xml-compliant user interface description languages. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2844:377–391, 2003.

[5] Jeffrey Nichols, Brad A Myers, Michael Higgins, Joseph Hughes, Thomas K Harris, Roni Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. 2002.

[6] Marco Winckler and Philippe Palanque. Models as representations for supporting the development of e-procedures. *Usablity in Government Systems User Experience Design for Citizens and Public Servants*, pages 301–315, 2012.

[7] Nikola Mitrovic, Carlos Bobed, and Eduardo Mena. A review of user interface description languages for mobile applications. 10 2016.

[8] Krzysztof Gajos and Daniel S Weld. Supple: Automatically generating user interfaces. 2004.

[9] Shankar R Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. Icrafter : A service framework for ubiquitous computing environments. 2001.

[10] Giulio Mori, Fabio Paternò, and Carmen Santoro. Tool support for designing nomadic applications. page 141, 2003.

[11] Simon Mayer, Andreas Tschofen, Anind K. Dey, and Friedemann Mattern. User interfaces for smart things - a generative approach with semantic interaction descriptions. *ACM Transactions on Computer-Human Interaction*, 21, 2014.

[12] Scrum guide. https://scrumguides.org/scrum-guide.html. Accessed: 2023-01-05.

[13] Linda Rising and Norman S. Janoff. Scrum software development process for small teams. *IEEE Software*, 17:26–32, 7 2000.

[14] Why spring. https://spring.io/why-spring. Accessed: 2023-08-14.

[15] Html advantages. https://www.geeksforgeeks.org/advantages-and-disadvanatges-of-html/. Accessed: 2023-08-14.