Deliverable 2
Project Design Document

C-Labs

Ashwath George	9733469
Nicholas Gignac	9128964
Robert Jakubowicz	6045707
Caroline Labbe	6320945
Brian Lam	1696785

For the course:
COMP354
Introduction to Software Engineering

Presented to:
Sutharsan Sivagnanam
July 24, 2013

Contents

List of Figures

List of Tables

1 Introduction

The C-Ker program, from C-Labs, is a Vessel Monitoring System, delivered during the course of COMP 354, where the teacher will be referred to as the customer. For this second delivery, this document brings up more accurate information about the Architectural Design of the software in more details, as well as provides Dynamic Design scenarios. We will also review the Cost Estimation originally done in Delivery 1, using the data and experience acquired during this delivery as well as the previous.

2 Architectural Design

In our project, we decided to use the MVP (Model-View-Presenter) architecture pattern instead of MVC. Both are very similar except for the workflow, and the Presenter and Controller which have different roles. MVP was selected over MVC because it reflects our workflow more accurately. In MVP, the user interacts with the View, which then sends the commands to the Presenter who returns the data for the View to display after having interacted with the Model. In MVC, the user interacts with the Controller, which depending on the users action, will interact with the Model and send the data to the View. Additional, if there is an update on the Model, the view can be notified. In MVP, the View never communicates with the Model, everything is done through the Presenter.

As seen in Figure 1 below, the system contains four subsystems: GUI, Authentication, Simulation and Parsing. The GUI's purpose it to display Widgets and let the user interact with the system. Authentication is used to validate the credentials of a user and only grants access to authorized users. Simulation performs the calculations on the vessels. Finally, Parsing is used to read scenario files.

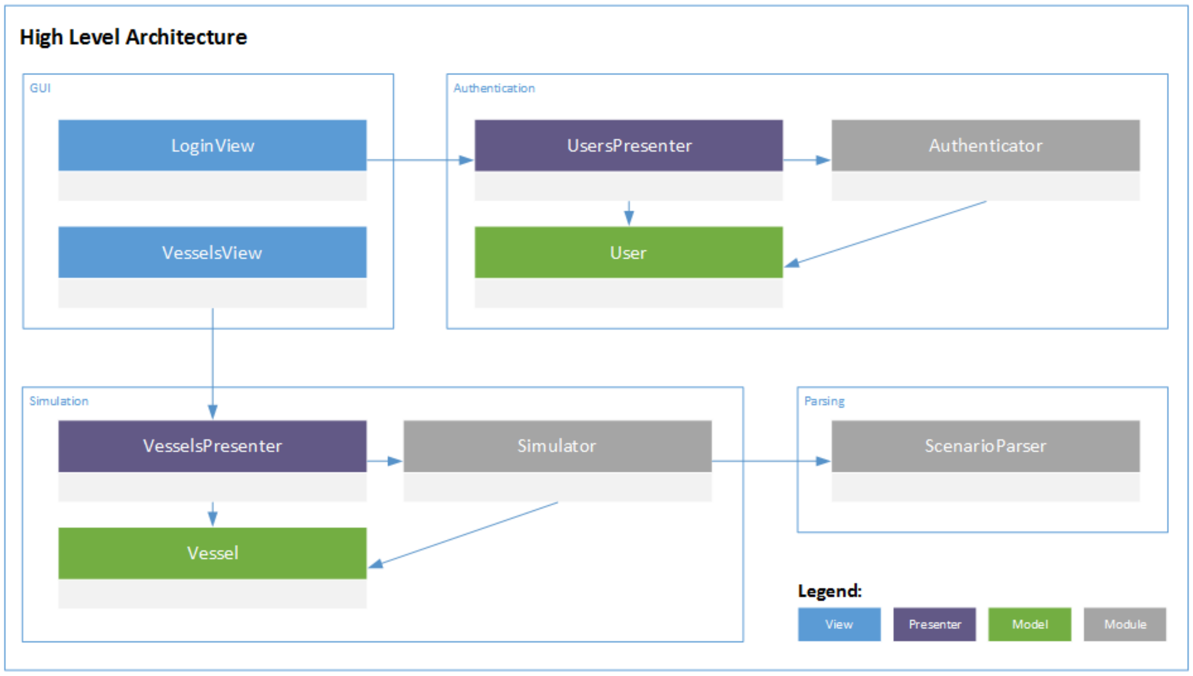


Figure 1: High Level Architecture

2.1 Architecture Diagram

The structure of the system is depicted using a 4+1 architectural view.

2.1.1 Logical View

Our system provides the user with four main functionalities: authentication, vessel viewing, vessel movement and alarms, as well as scenario file parsing. These functionalities are provided using components illustrated in the class diagram in Figure 2.

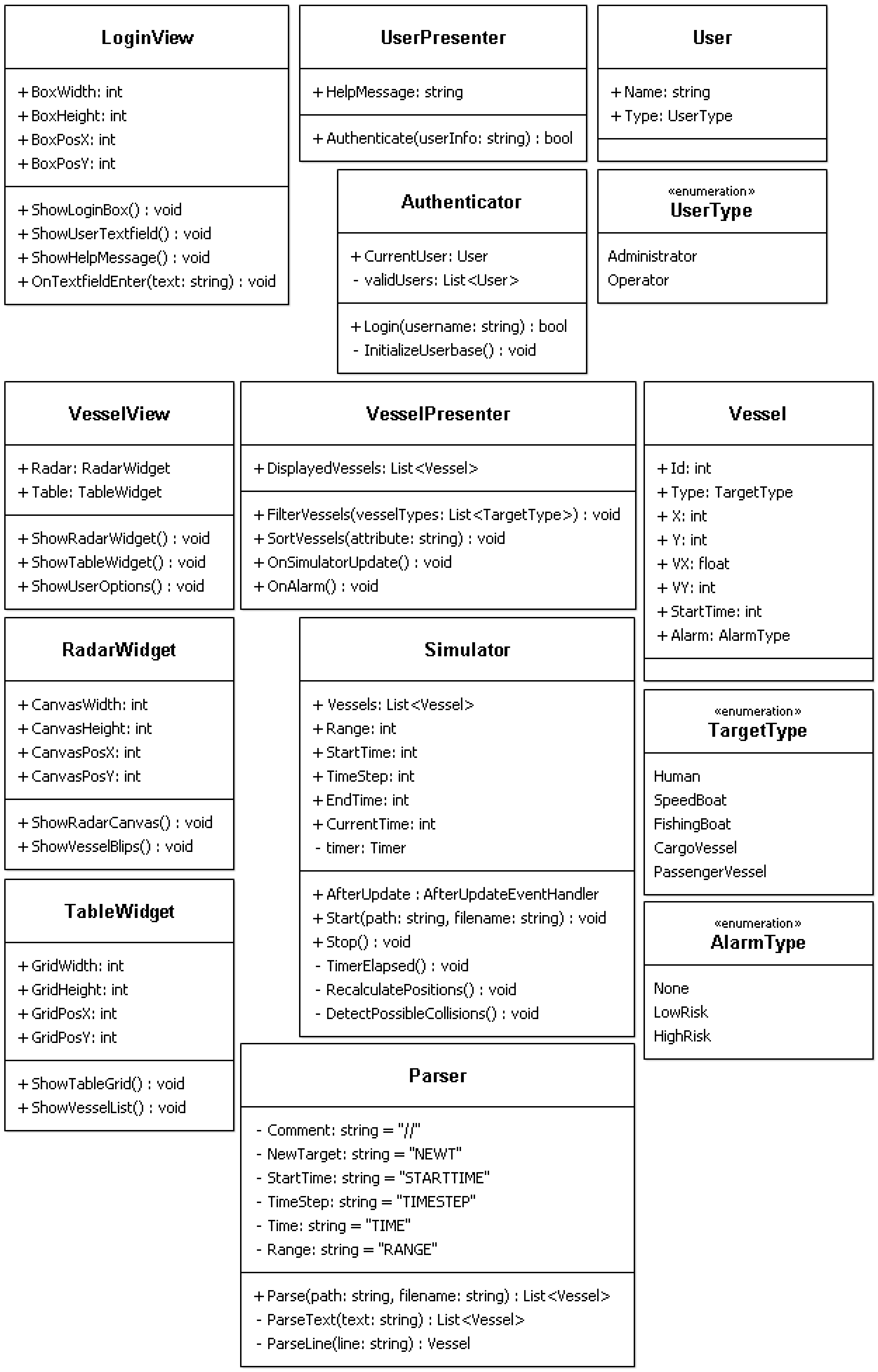


Figure 2: System Class diagram

Authentication allows users to login and access the system. As shown in Figure 3, the Authenticate() function is called with the username. If the specific username is allowed access, the Authenticate() returns true and false otherwise. Different user types have different priviledges when viewing the vessels.

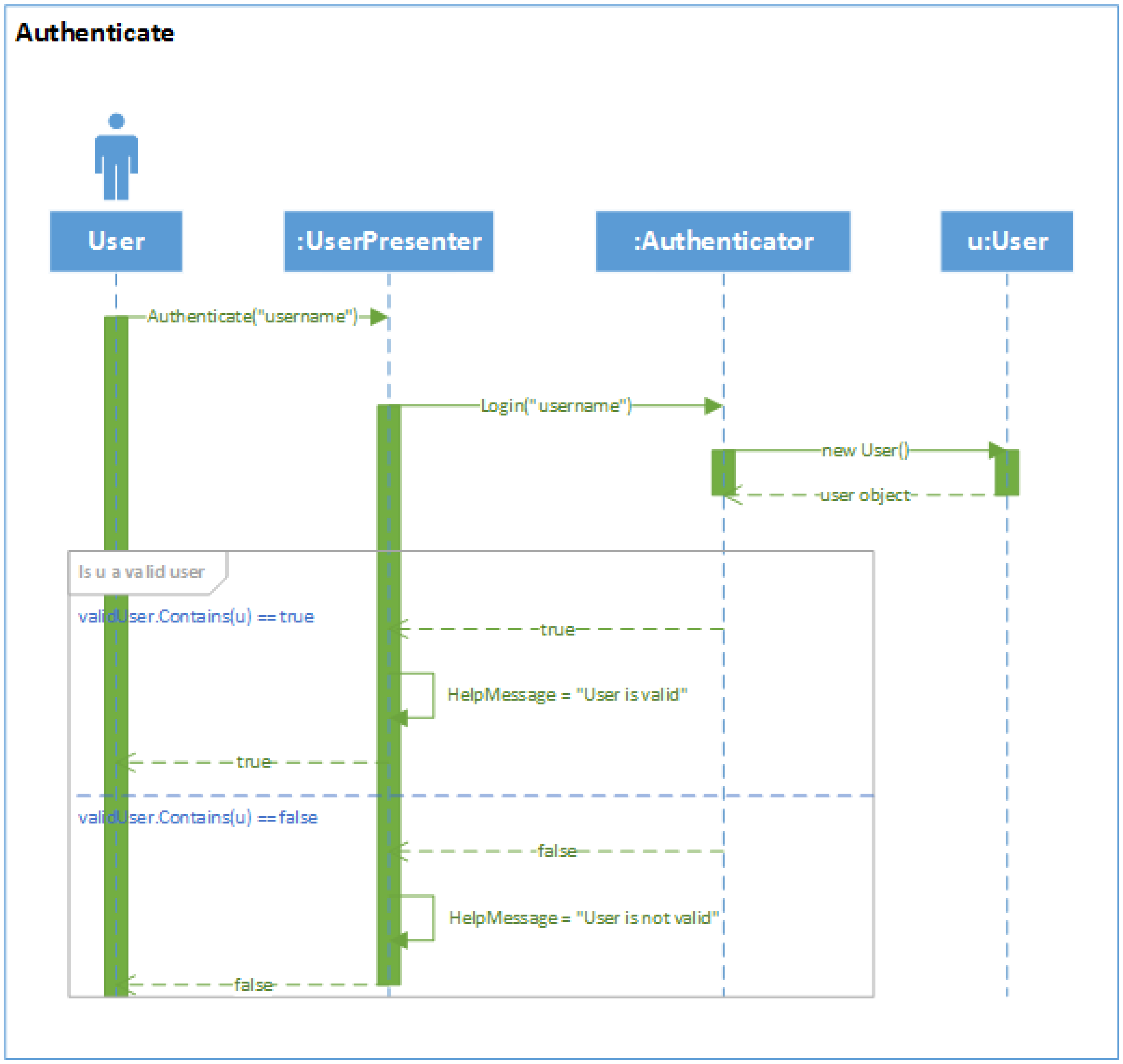


Figure 3: Authenticate Sequence Diagram

Vessel viewing allows users to observe vessels in both table list form and radar map form. Furthermore, users can filter and sort vessels that are displayed. As shown in

Figure 4, filtering is performed by the VesselsPresenter by calling the function GetVessels() and passing the type to filter on. It returns a filtered list of vessels which is then used to update the table and radar.

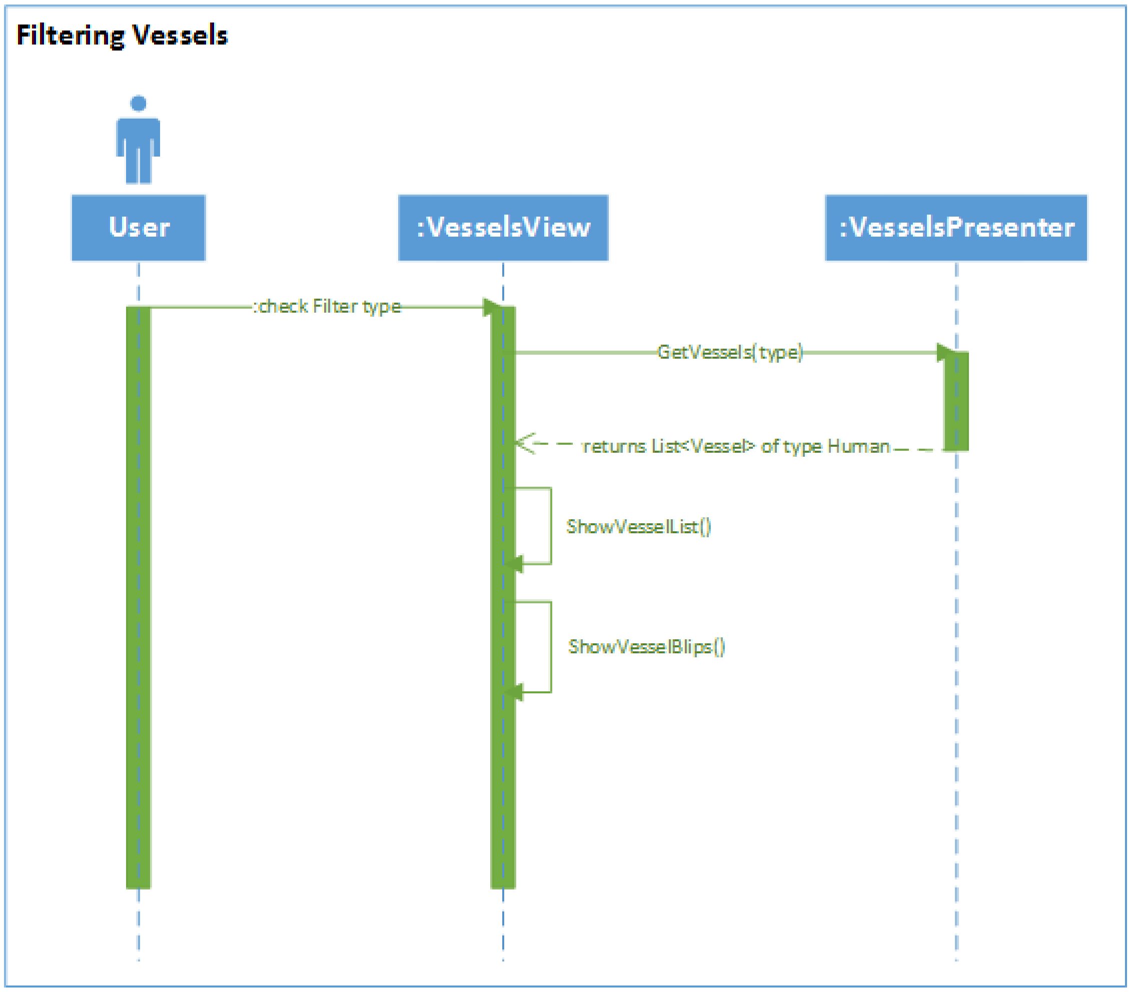


Figure 4: Filtering Sequence Diagram

As shown in Figure 5, sorting is performed by the VesselsPresenter by calling the function SortVessels() and passing the attribute to sort on. It returns a sorted list of vessels which is then used by VesselView.

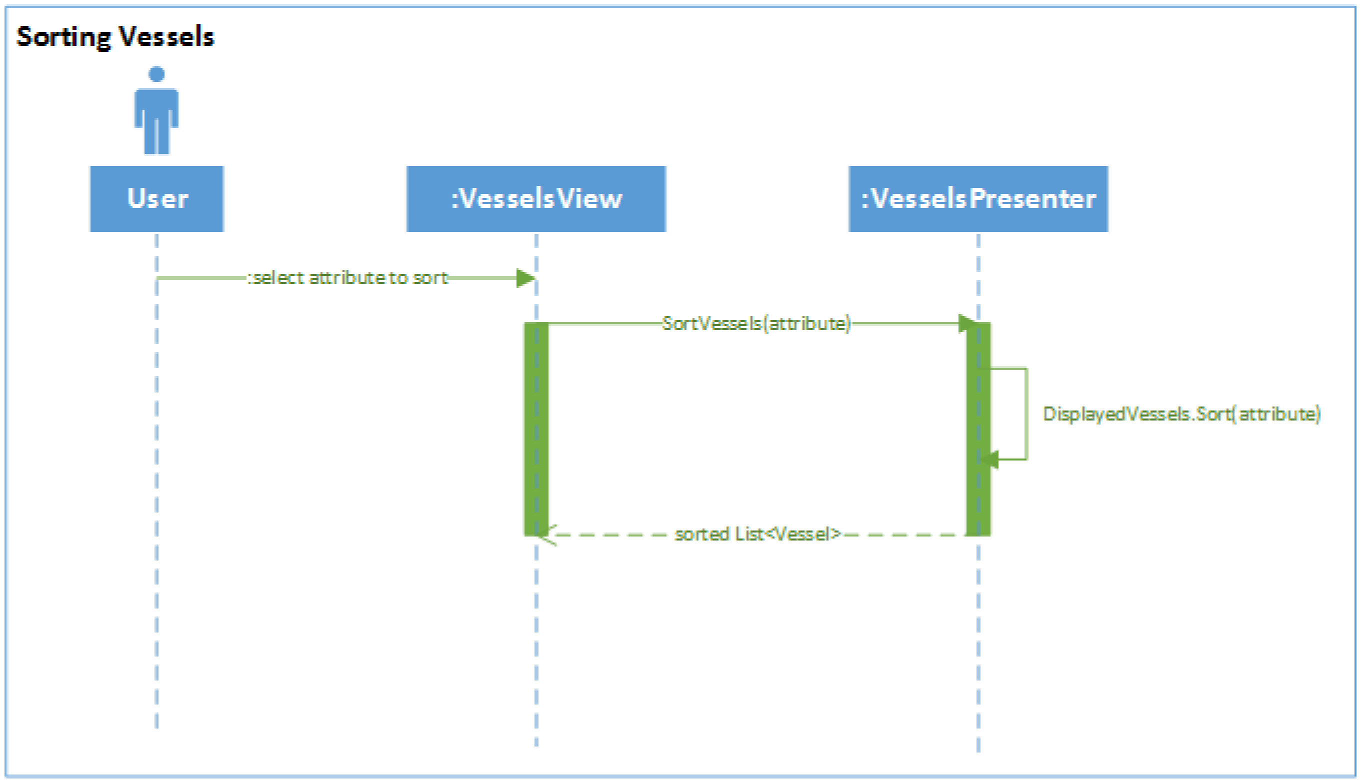


Figure 5: Sorting Sequence Diagram

Vessel movement and alarms allows users to receive dynamic updates of the vessels current statuses. As illustrated in Figure 6, the Simulator loops until the EndTime is reached. At each iteration, it recalculates the position and notifies the VesselPresenter that the positions have been updated by raising an event.

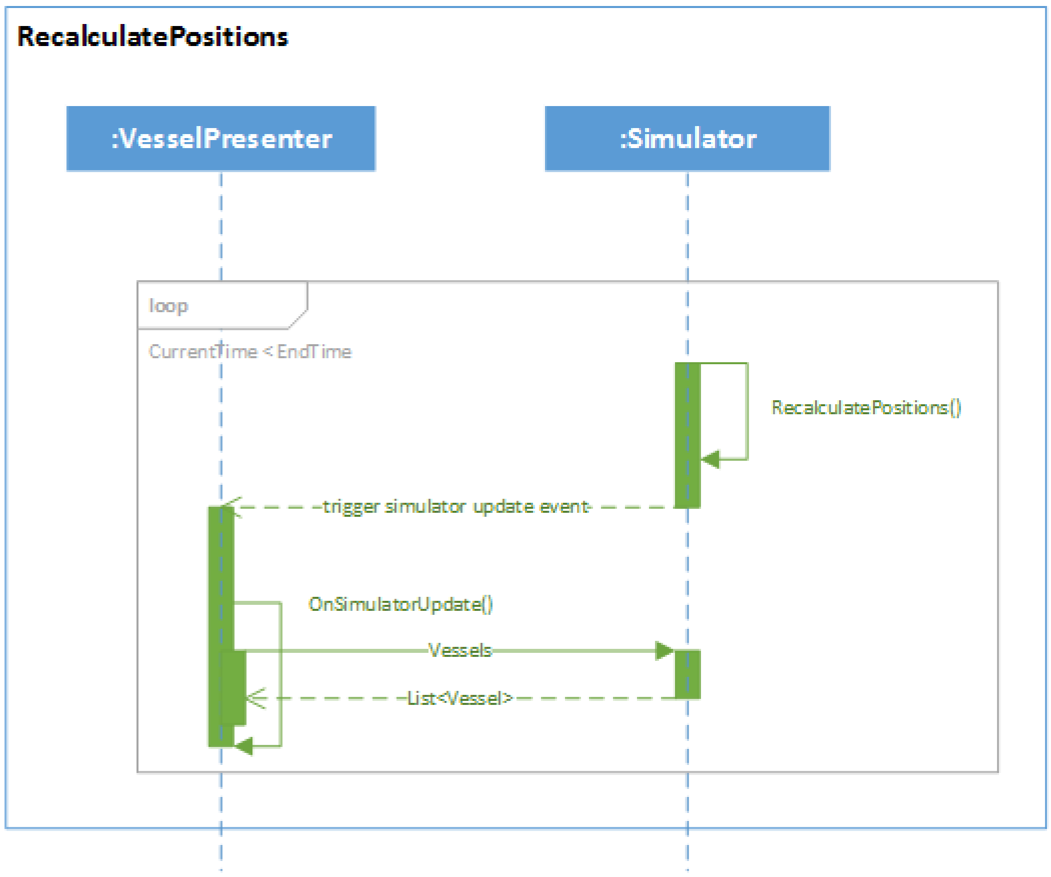


Figure 6: Recalculate Positions Sequence Diagram

Parsing allows scenario files to be read in order to load simulation properties and vessel initial data. As seen in Figure 7, the function Parse() is call and passed the path and the filename of the scenario file. First the text is parsed, and then each line of the text. A list of vessels is returned to the Simulator.

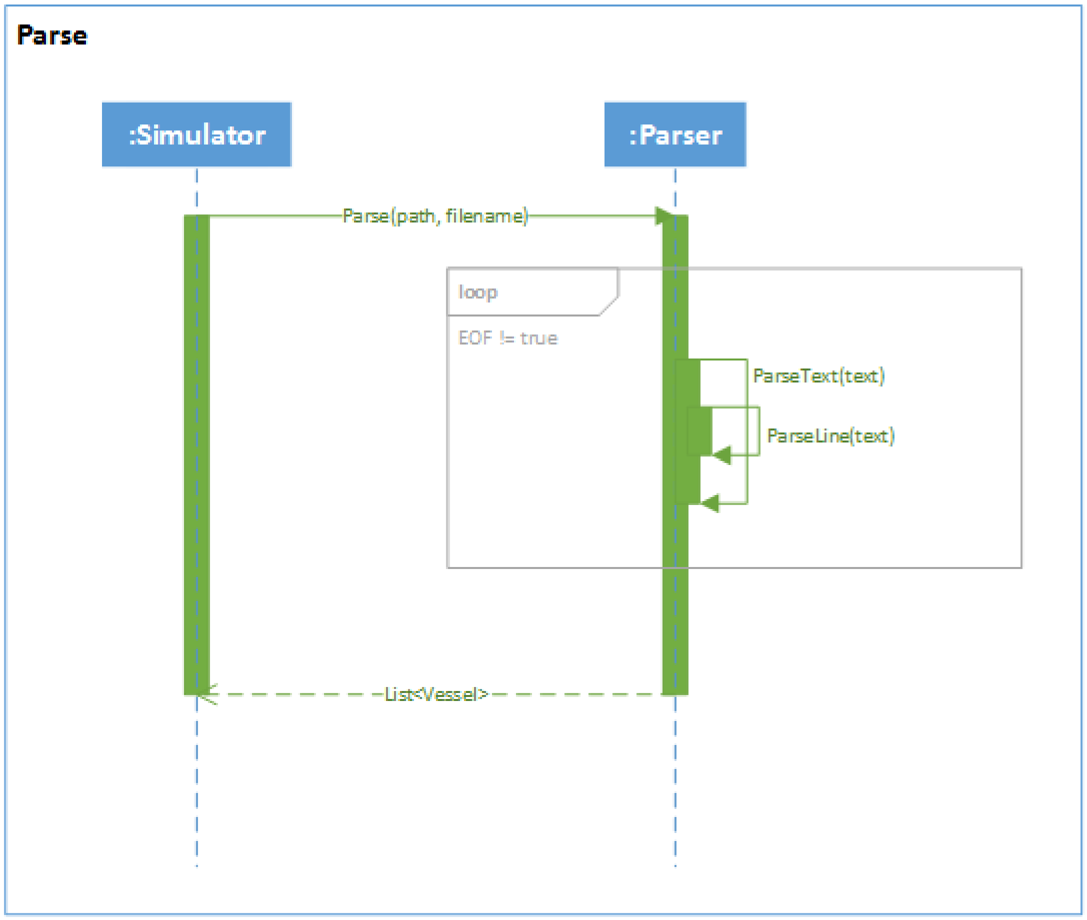


Figure 7: Parse Sequence Diagram

2.1.2 Development View

The static organization of the C-Ker Vessel Monitoring System in its development environment is demonstrated through the use of a Component Diagram. As shown in Figure 8, the decomposition of the C-Ker code base into four distinct code components allows for a clearer view of their implementations, including the interdependencies between the modules contained within. This serves as a basis for requirement allocation and the allocation of work to the programmers, while also facilitating a comprehensive estimation the cost evaluation and planning stages of the project.

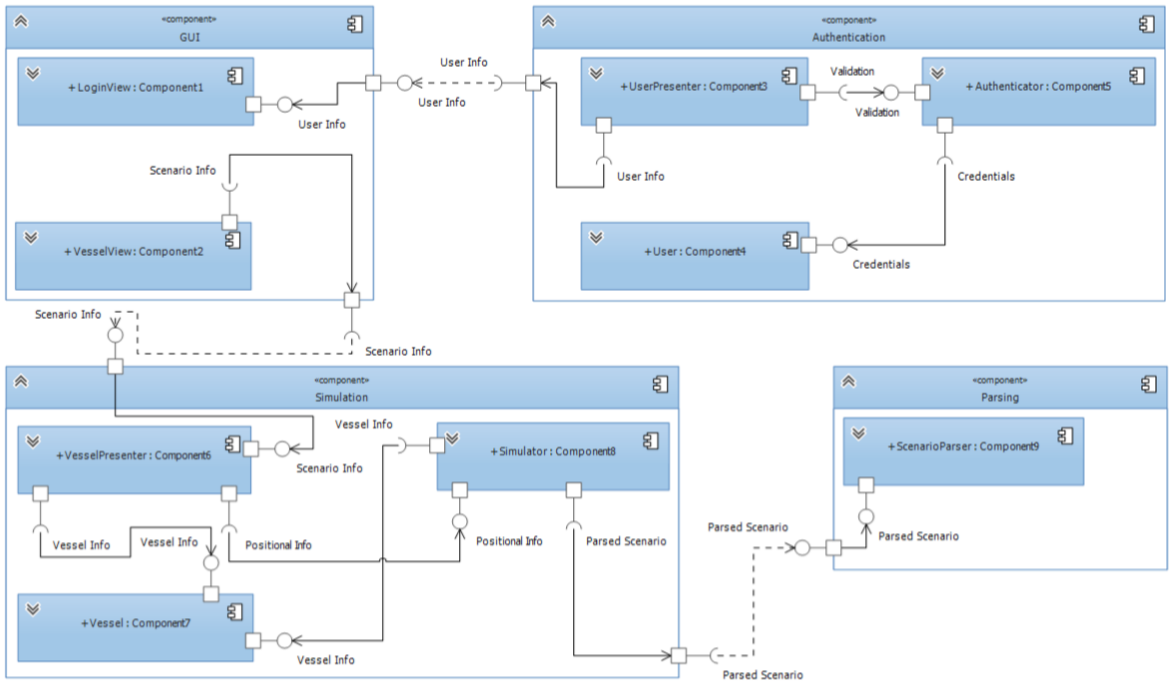


Figure 8: Component Diagram

2.1.3 Process View

The concurrency and synchronization aspects of the design of the C-Ker System can be best characterized through an Activity Diagram. Figure 9 depicts the runtime behaviour of the system as a whole, highlighting the mapping of logical components to processes. Figure 10, on the other hand, focuses on the manner in which the system handles the process of triggering alarms by modelling the concurrent behaviour of the Simulation subsystem.

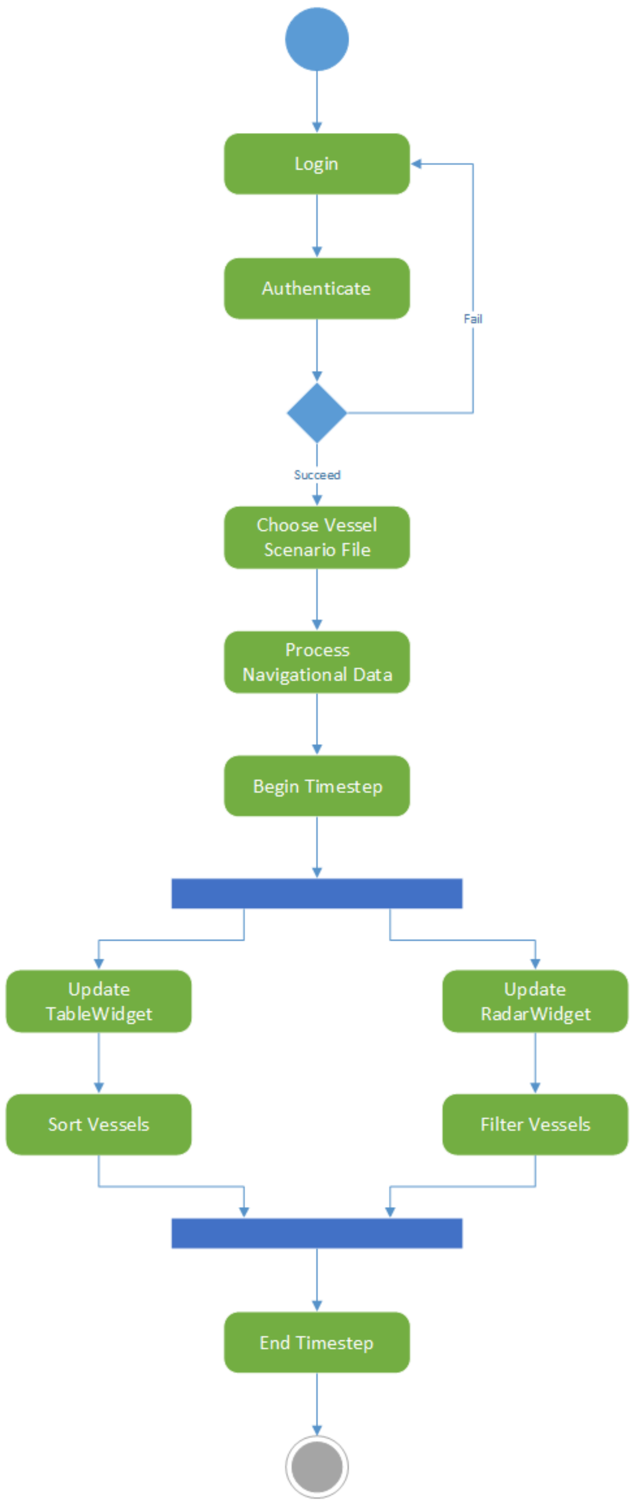


Figure 9: Main Activity diagram

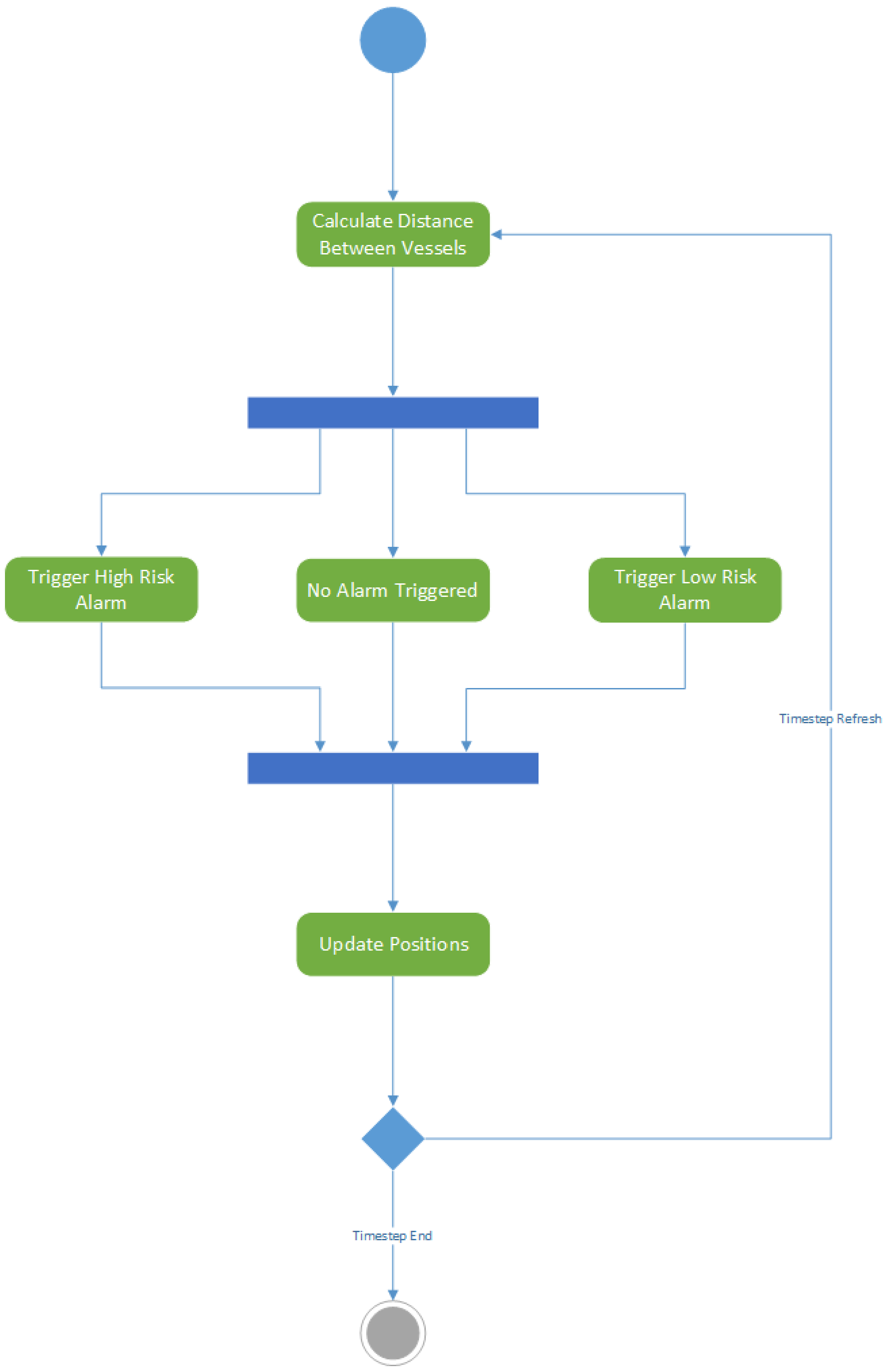


Figure 10: Alarm Activity Diagram

2.1.4 Physical View

This view does not apply to our project.

2.1.5 Scenarios

The architecture is further illustrated using a set of use cases as shown in Figure 11.

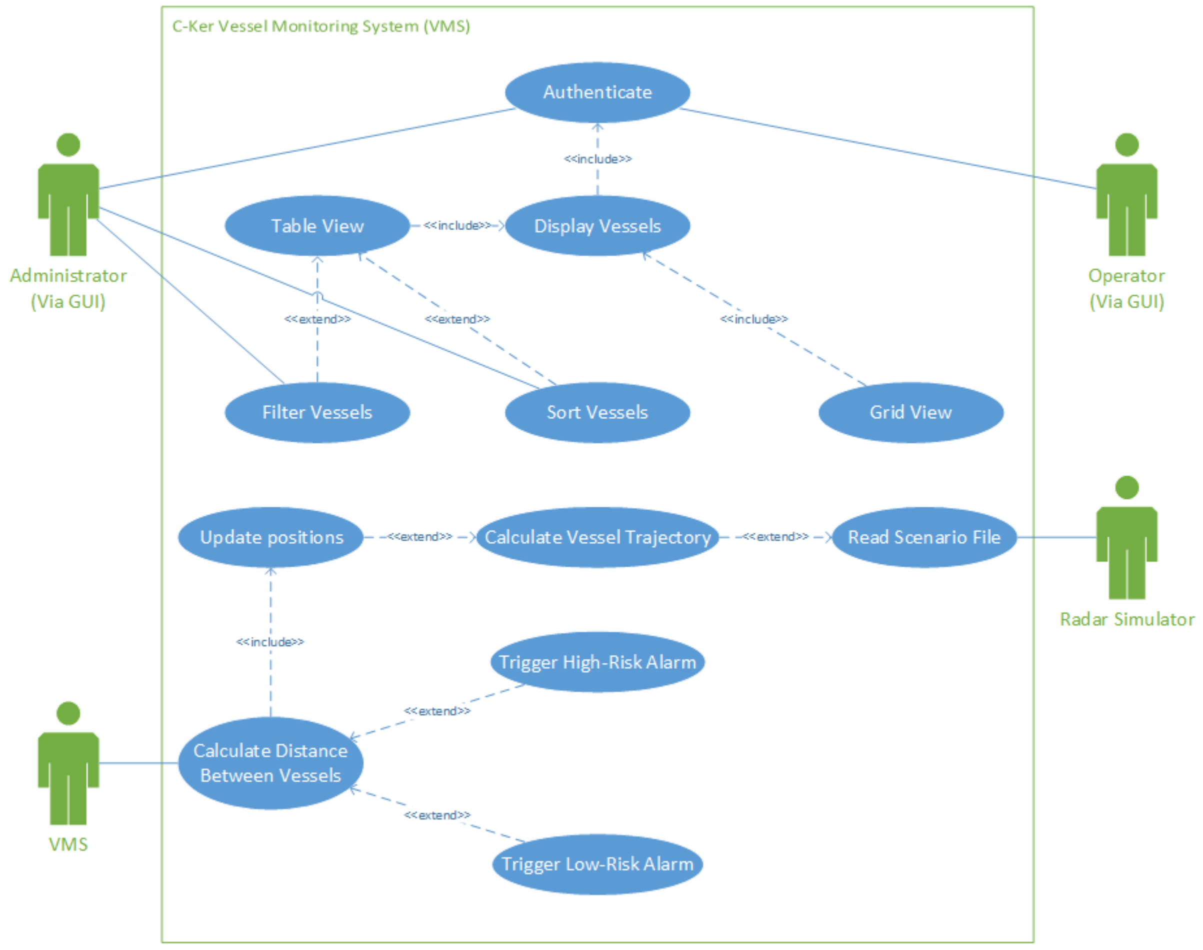


Figure 11: Use Case diagram

Scenarios

Name	Listing Vessels in a Table
Actors	Any user, TableView, Presenter
Goals	Allow users to view vessel information listed in a table
Pre-conditions	Vessel information must be loaded and updated correctly
Summary	Information about each vessel is displayed in a table format
Related Use Cases	Updating Vessel Positions
Steps	- The TableView retrieves vessel information from the Presenter - The application displays vessel information in a table on screen - The user views vessel information as desired
Post-conditions	The vessel information are listed in a table on screen
Difficulty	Low
Importance	High

Table 1: Listing Vessels in a Table

Name	Filtering Vessels by Type
Actors	User with admin privileges
Goals	Allow users to view or hide certain vessel types only
Pre-conditions	- Vessel information must be loaded and listed correctly - User must be authenticated as an administrator
Summary	The displayed vessel information can be filtered by vessel type
Related Use Cases	Listing Vessels in a Table
Steps	- The user selects vessel types to be filtered through the interface - The application shows only the wanted vessel types on screen - The user views vessel information as desired
Post-conditions	The vessels displayed match the filtering options
Difficulty	Medium
Importance	High

Table 2: Filtering Vessels by Type

Name	Sorting Vessels by Data
Actors	User with admin privileges
Goals	Allow users to organize and view vessels with more flexibility
Pre-conditions	- Vessel information must be loaded and listed correctly - User must be authenticated as an administrator
Summary	The displayed vessel information can be sorted according to selected data
Related Use Cases	Listing Vessels in a Table
Steps	- The user selects a data type through the application interface - The application sorts the listing according to the selected data - The user views vessel information as desired
Post-conditions	The vessels displayed match the filtering options
Difficulty	Medium
Importance	High

Table 3: Sorting Vessels by Data

Name	Displaying Vessels on Radar
Actors	Any user, RadarView, Presenter
Goals	Give users a top-down visualization of the simulation in 2D space
Pre-conditions	Vessel information must be updated continuously and correctly
Summary	Vessels are represented graphically through blips on a radar display
Related Use Cases	Updating Vessel Positions
Steps	- The RadarView retrieves vessel information from the Presenter - The application draws the radar, with blips representing vessel positions - The radar is redrawn at regular intervals to reflect updates in simulation - The user views the map and vessels as desiredn
Post-conditions	The vessel information are listed in a table on screen
Difficulty	Medium
Importance	High

Table 4: Displaying Vessels on Radar

Name	Loading Vessel Information
Actors	Simulator
Goals	Load vessel information into memory from a scenario file
Pre-conditions	Scenario file must exist and be formatted correctly
Summary	The radar simulator reads information on simulation and vessel properties
Related Use Cases	None
Steps	- A specific scenario file is provided - The simulator reads the file, parses the relevant info, and stores it
Post-conditions	All simulation and vessel data are loaded and ready to be simulated
Difficulty	Low
Importance	High

Table 5: Loading Vessel Information

Name	Updating Vessel Positions
Actors	Simulator
Goals	Simulate vessel movement at regular intervals
Pre-conditions	Vessel information must be loaded correctly
Summary	The radar simulator plots vessel trajectories by performing the necessary physics calculations using the information parsed from the corresponding scenario file.
Related Use Cases	Loading Vessel Information
Steps	- The simulator begins with the initial position of the vessel - The vessels subsequent positions are calculated over time, based on its velocity along the coordinate axes
Post-conditions	Vessel positions are updated correctly
Difficulty	Low
Importance	High

Table 6: Updating Vessel Positions

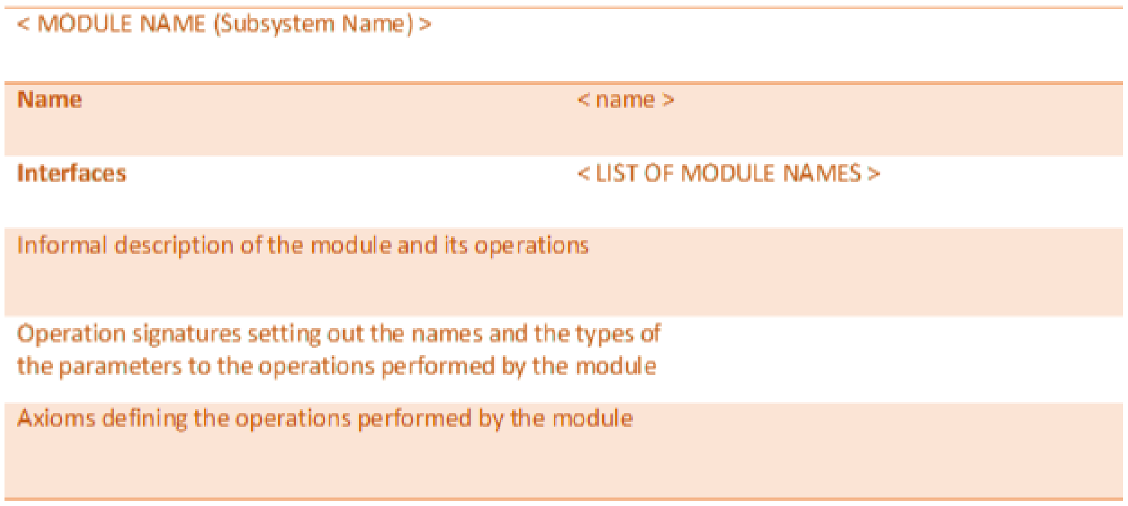
Name	Triggering Risk Alarms
Actors	Simulator
Goals	Trigger alarms to warn users when vessels get too close to each other
Pre-conditions	Vessel information must be updated continuously and correctly
Summary	The radar simulator calculates distances between vessels, and triggers specific alarms under certain conditions.
Related Use Cases	Updating Vessel Positions
Steps	- The simulator calculates distances between vessels after every update - If the distance between any two vessels is between 200m and 50m, the simulator triggers a Low Risk Alarm for those vessels - If the distance between any two vessels is less than 50m, the simulator triggers a High Risk Alarm for those vessels
Post-conditions	Alarms are triggered if conditions are met
Difficulty	Low
Importance	High

Table 7: Triggering Risk Alarms

2.2 Subsystem Interfaces Specifications

The purpose of this Subsystem Interface Specification is to describe the messages and function calls that are exchanged between the various modules of the C-Ker Vessel Monitoring System. It is intended to specify the externally observable behavior of a modules access routines, including input/output relationships and parameters that could be considered either valid or invalid.

The Figures in sections 2.2.1 and 2.2.2 provide specifications for interfaces within the GUI subsystem, sections 2.2.3 to 2.2.6 belong to the Authentication subsystem, sections 2.2.7 and 2.2.8 belong to the Simulation subsystem, and finally section 2.2.9 belongs to the Parsing subsystem.



< MODULE NAME (Subsystem Name) >	
Name	< name >
Interfaces	< LIST OF MODULE NAMES >
Informal description of the module and its operations	
Operation signatures setting out the names and the types of the parameters to the operations performed by the module	
Axioms defining the operations performed by the module	

Figure 12: Subsystem Interface Specification template

The structure of the Subsystem Interface Specification is shown in Figure 12. The body of the specification has four components:

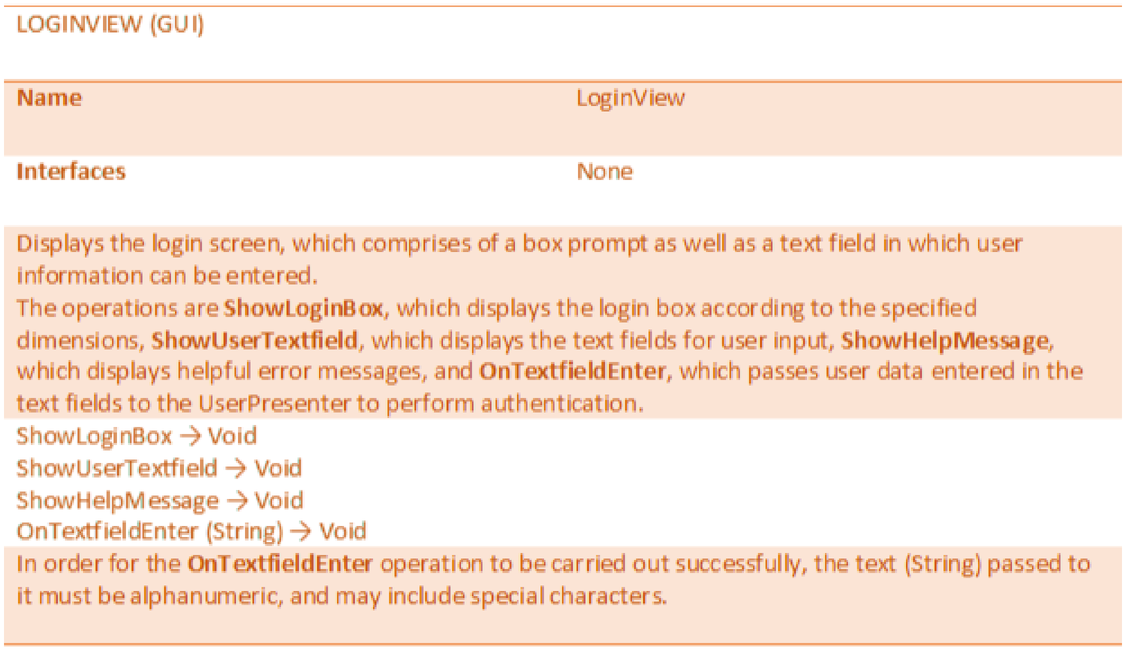
1. An introduction that declares the name of the module being specified. The introduction may also include an Interfaces declaration, which lists the other subsystems with which the current module interfaces, if any.

2. A description, which informally describes the module and the operations (functions or methods/member functions) it performs. This allows for easier interpretation of the formal specification.

3. The signature part, which defines the syntax of the interface between the modules. The names of the operations that are defined, the data types of their arguments, parameters and outputs are described in the signature.

4. The axioms part, which characterizes the behavior of the operations by defining the valid range of inputs required for the operation to be carried out successfully.

2.2.1 LoginView Module: GUI Subsystem



LOGINVIEW (GUI)

Name	LoginView
Interfaces	None

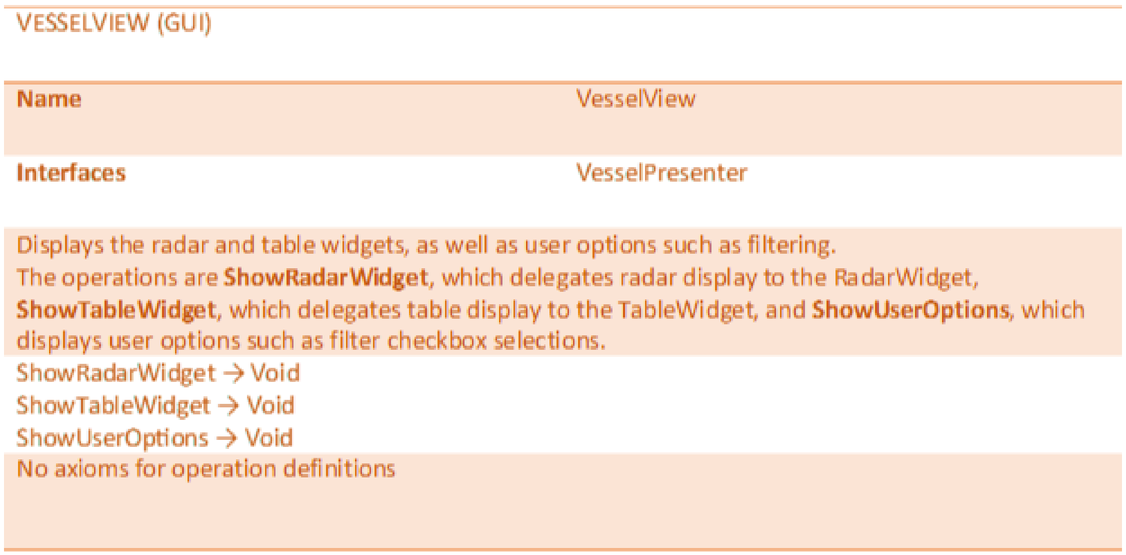
Displays the login screen, which comprises of a box prompt as well as a text field in which user information can be entered.
The operations are **ShowLoginBox**, which displays the login box according to the specified dimensions, **ShowUserTextfield**, which displays the text fields for user input, **ShowHelpMessage**, which displays helpful error messages, and **OnTextfieldEnter**, which passes user data entered in the text fields to the UserPresenter to perform authentication.

ShowLoginBox → Void
ShowUserTextfield → Void
ShowHelpMessage → Void
OnTextfieldEnter (String) → Void

In order for the **OnTextfieldEnter** operation to be carried out successfully, the text (String) passed to it must be alphanumeric, and may include special characters.

Figure 13: LoginView

2.2.2 VesselView Module: GUI Subsystem



VESSELVIEW (GUI)

Name	VesselView
Interfaces	VesselPresenter

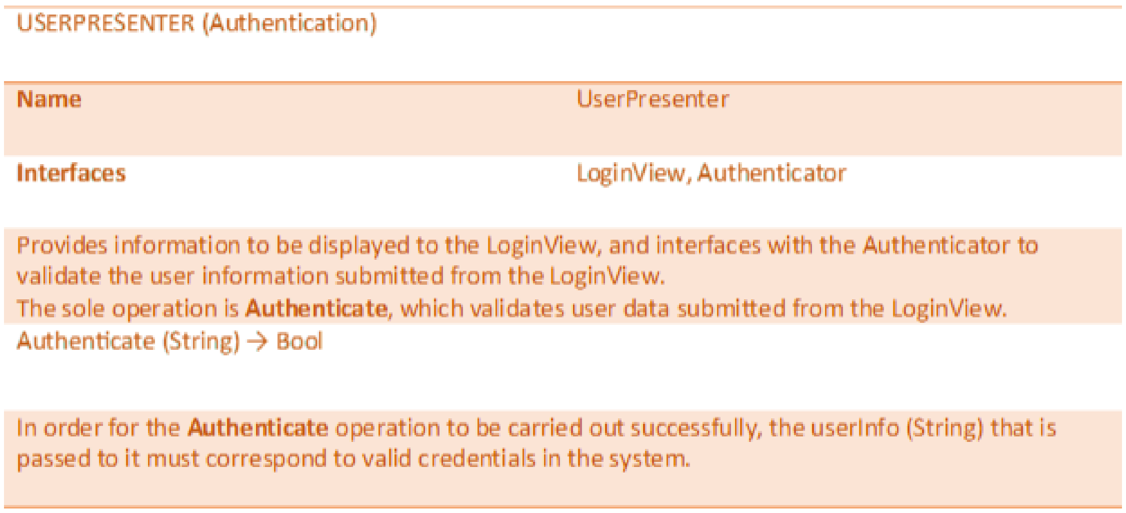
Displays the radar and table widgets, as well as user options such as filtering.
The operations are **ShowRadarWidget**, which delegates radar display to the RadarWidget, **ShowTableWidget**, which delegates table display to the TableWidget, and **ShowUserOptions**, which displays user options such as filter checkbox selections.

ShowRadarWidget → Void
ShowTableWidget → Void
ShowUserOptions → Void

No axioms for operation definitions

Figure 14: VesselView

2.2.3 UserPresenter Module: Authentication Subsystem



USERPRESENTER (Authentication)

Name	UserPresenter
Interfaces	LoginView, Authenticator

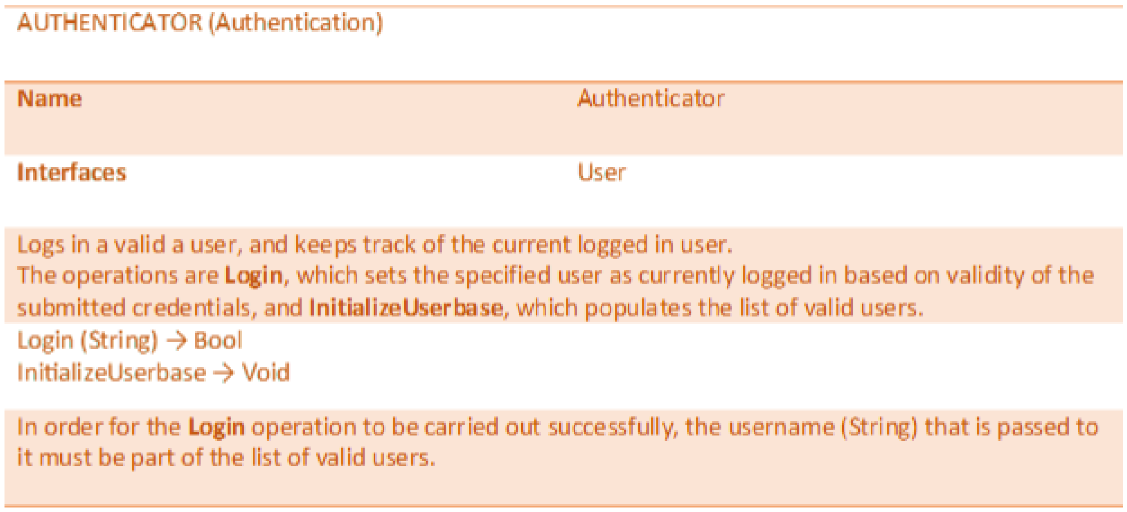
Provides information to be displayed to the LoginView, and interfaces with the Authenticator to validate the user information submitted from the LoginView.
The sole operation is **Authenticate**, which validates user data submitted from the LoginView.

Authenticate (String) → Bool

In order for the **Authenticate** operation to be carried out successfully, the userInfo (String) that is passed to it must correspond to valid credentials in the system.

Figure 15: UserPresenter Module

2.2.4 Authenticator Module: Authentication Subsystem



AUTHENTICATOR (Authentication)

Name	Authenticator
Interfaces	User

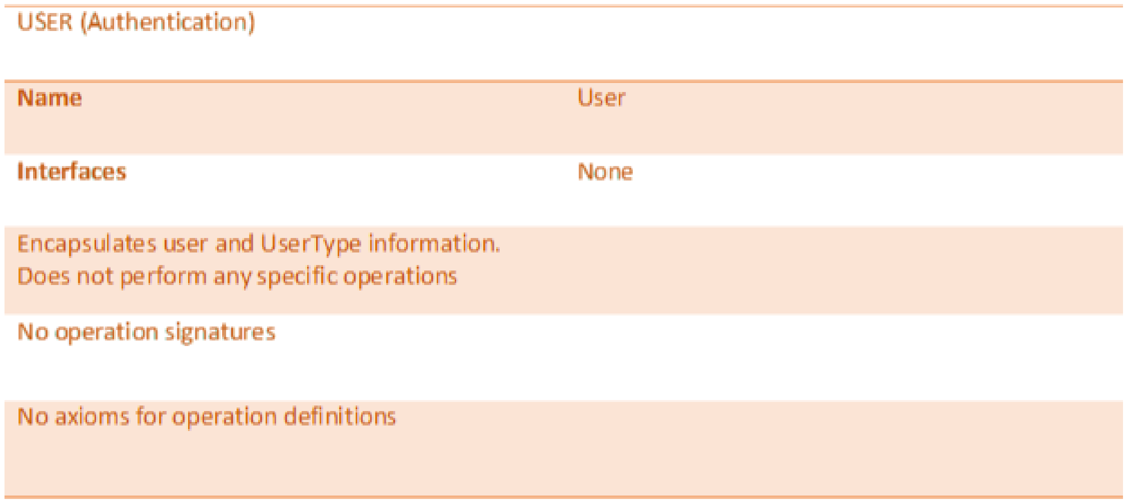
Logs in a valid a user, and keeps track of the current logged in user.
The operations are **Login**, which sets the specified user as currently logged in based on validity of the submitted credentials, and **InitializeUserbase**, which populates the list of valid users.

Login (String) → Bool
InitializeUserbase → Void

In order for the **Login** operation to be carried out successfully, the username (String) that is passed to it must be part of the list of valid users.

Figure 16: Authenticator Module

2.2.5 User Module: Authentication Subsystem



USER (Authentication)

Name	User
Interfaces	None

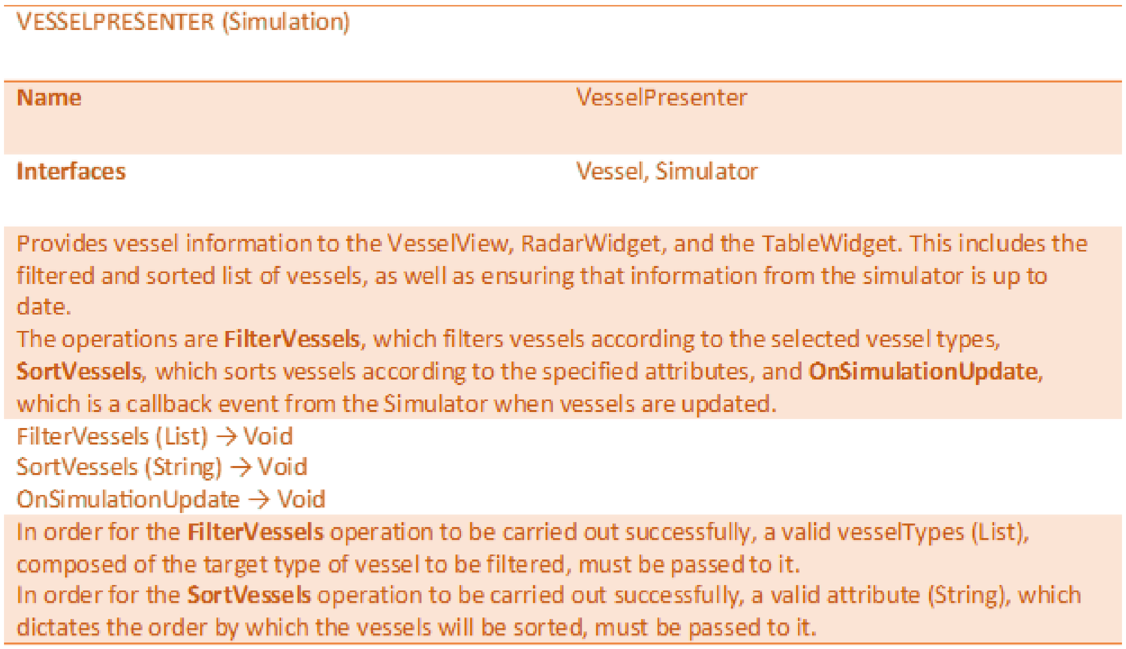
Encapsulates user and UserType information.
Does not perform any specific operations

No operation signatures

No axioms for operation definitions

Figure 17: User Module

2.2.6 VesselPresenter Module: Simulation Subsystem



VESSELPRESENTER (Simulation)

Name	VesselPresenter
Interfaces	Vessel, Simulator

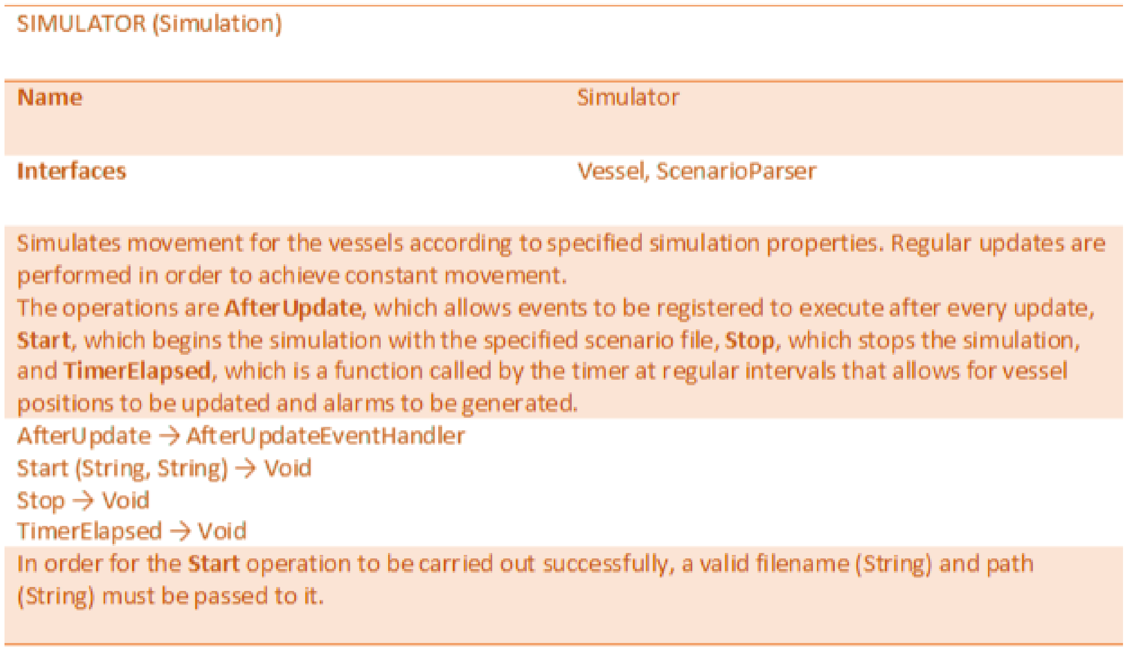
Provides vessel information to the VesselView, RadarWidget, and the TableWidget. This includes the filtered and sorted list of vessels, as well as ensuring that information from the simulator is up to date.
The operations are **FilterVessels**, which filters vessels according to the selected vessel types, **SortVessels**, which sorts vessels according to the specified attributes, and **OnSimulationUpdate**, which is a callback event from the Simulator when vessels are updated.

FilterVessels (List) → Void
SortVessels (String) → Void
OnSimulationUpdate → Void

In order for the **FilterVessels** operation to be carried out successfully, a valid vesselTypes (List), composed of the target type of vessel to be filtered, must be passed to it.
In order for the **SortVessels** operation to be carried out successfully, a valid attribute (String), which dictates the order by which the vessels will be sorted, must be passed to it.

Figure 18: VesselPresenter Module

2.2.7 Simulator Module: Simulation Subsystem



SIMULATOR (Simulation)

Name	Simulator
Interfaces	Vessel, ScenarioParser

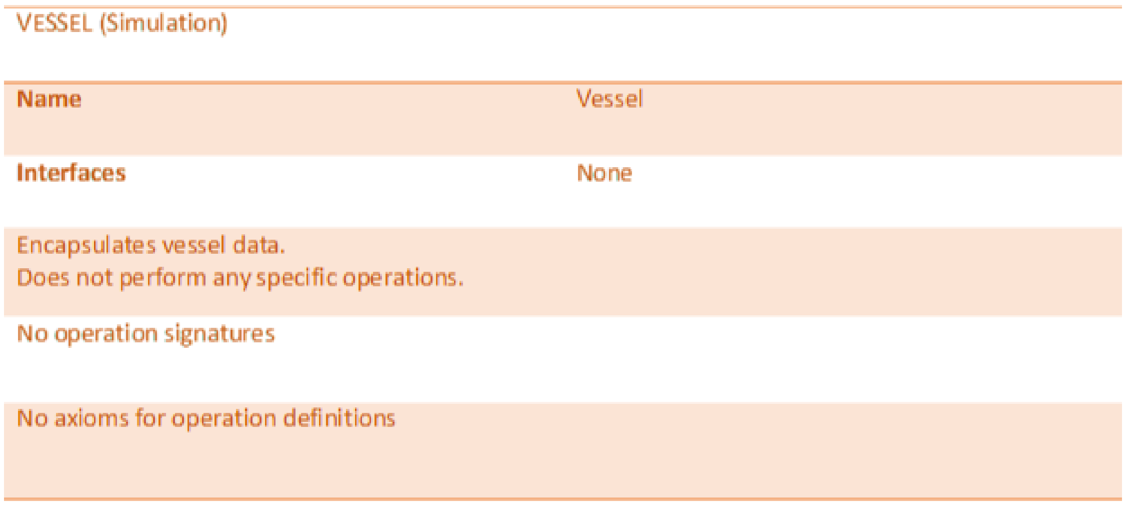
Simulates movement for the vessels according to specified simulation properties. Regular updates are performed in order to achieve constant movement.
The operations are **AfterUpdate**, which allows events to be registered to execute after every update, **Start**, which begins the simulation with the specified scenario file, **Stop**, which stops the simulation, and **TimerElapsed**, which is a function called by the timer at regular intervals that allows for vessel positions to be updated and alarms to be generated.

AfterUpdate → AfterUpdateEventHandler
Start (String, String) → Void
Stop → Void
TimerElapsed → Void

In order for the **Start** operation to be carried out successfully, a valid filename (String) and path (String) must be passed to it.

Figure 19: Simulator Module

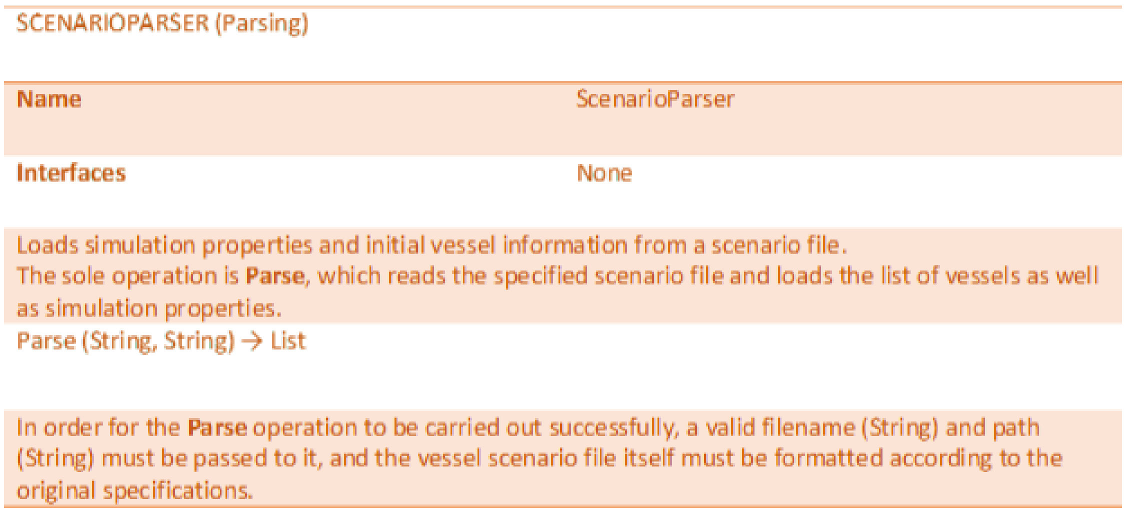
2.2.8 Vessel Module: Simulation Subsystem



VESSEL (Simulation)	
Name	Vessel
Interfaces	None
Encapsulates vessel data. Does not perform any specific operations.	
No operation signatures	
No axioms for operation definitions	

Figure 20: Vessel Module

2.2.9 ScenarioParser Module: Parsing Subsystem



SCENARIOPARSER (Parsing)	
Name	ScenarioParser
Interfaces	None
Loads simulation properties and initial vessel information from a scenario file. The sole operation is **Parse**, which reads the specified scenario file and loads the list of vessels as well as simulation properties.	
Parse (String, String) → List	
In order for the **Parse** operation to be carried out successfully, a valid filename (String) and path (String) must be passed to it, and the vessel scenario file itself must be formatted according to the original specifications.	

Figure 21: ScenarioParser Module

3 Detailed Design

The system is divided into four subsystems: GUI, Authentication, Simulation, and Parsing. They each handle independent responsibilities which makes the system easier to understand and to maintain.

3.1 GUI Subsystems

The GUI subsystem takes care of displaying information on screen; it also provides the user with an interface to use the application.

3.1.1 Detailed Design Diagram

This subsystem consists of the LoginView, the VesselView, the RadarWidget, and the TableWidget classes as shown in Figure 22. This design allows the GUI elements to be modularized and isolated.

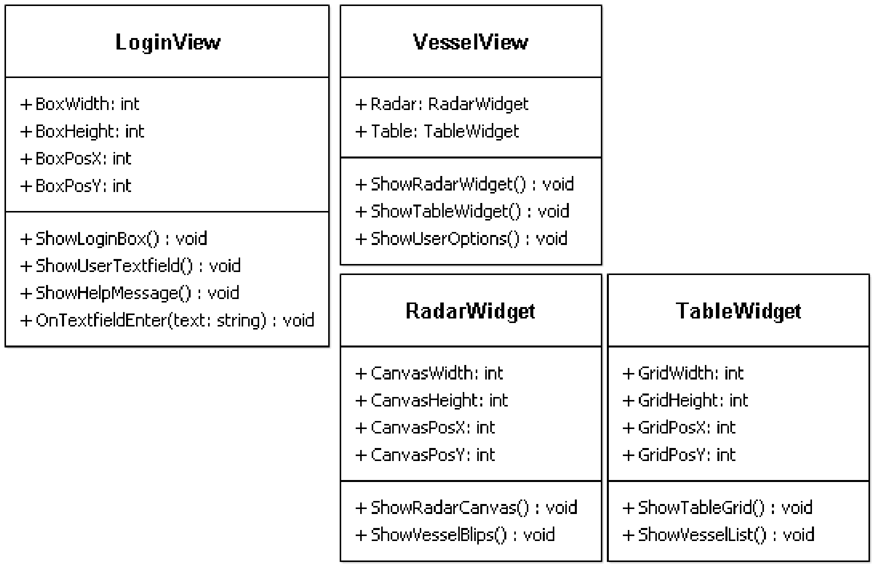


Figure 22: GUI Subsystem Class diagram

3.1.2 Unit Descriptions

LoginView

LoginViews purpose is to display the login screen; this includes a box prompt as well as a text field in which user information can be entered.

Name	Description
BoxWidth BoxHeight BoxPosX BoxPosY	Defines the dimensions of the login prompt.
ShowLoginBox	Displays the login box according to dimensions.
ShowUserTextfield	Displays the text field for the user information to be entered.
ShowHelpMessage	Displays any messages which help inform the user; the messages are retrieved from UserPresenter.
OnTextfieldEnter	Callback when the user submits text in the text field; the data is to be passed to the UserPresenter to perform authentication.

Table 8: LoginView Attributes and Methods

VesselView

VesselViews purpose is to display the radar, the table, as well as user options such as filtering selection.

Name	Description
Radar	Reference to the RadarWidget.
Table	Reference to the TableWidget.
ShowRadarWidget	Delegate radar display to the RadarWidget.
ShowTableWidget	Delegate table display to the TableWidget.
ShowUserOptions	Displays user options such as filter checkbox selections.

Table 9: VesselView Attributes and Methods

RadarWidget

RadarWidgets purpose is to display the radar, which illustrates visually the location of the vessels.

Name	Description
CanvasWidth CanvasHeight CanvasPosX CanvasPosY	Defines the dimensions of the radar background canvas.
ShowRadarCanvas	Displays the radar background. There are many possibilities such as displaying a blue background to represent the sea or having green grid with circles to imitate the radar look.
ShowVesselBlips	Displays the vessels on the radar at the correct positions. This will involve converting the position units to pixel scale. The list of vessels to show is retrieved from the VesselPresenter.

Table 10: RadarWidget Attributes and Methods

TableWidget

TableWidgets purpose is to display the table, in which the vessel information will be listed.

Name	Description
GridWidth GridHeight GridPosX GridPosY	Defines the dimensions of the table grid.
ShowTableGrid	Displays the table grid. This involves setting up the columns to match the vessel attributes as well as reserving enough width space for each column.
ShowVesselList	Displays the vessel information in the table. Each vessel would be displayed on a separate row of the table. The list of vessels to show is retrieved from the VesselPresenter.

Table 11: TableWidget Attributes and Methods

3.2 Authentication Subsystem

This subsystem handles the actual logic for authenticating user information.

3.2.1 Detailed Design Diagram

This subsystem consists of the UserPresenter, the Authenticator and the User classes as shown in Figure 23.

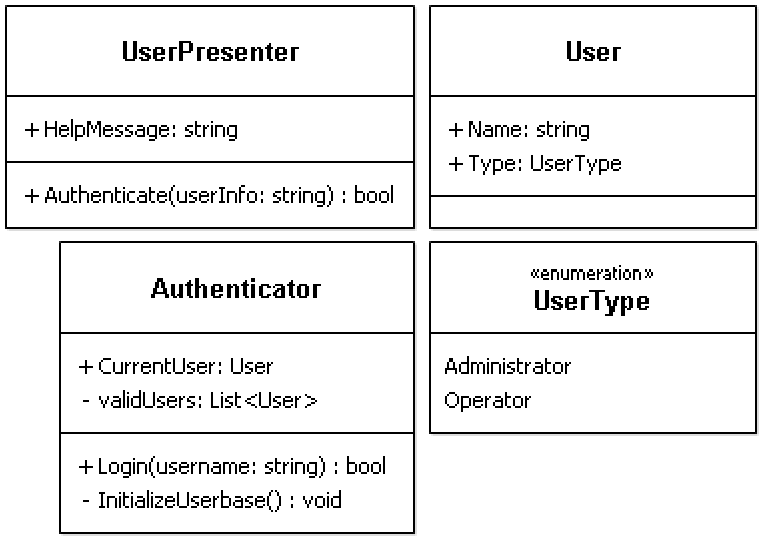


Figure 23: Authentification Subsystem Class diagram

3.2.2 Unit Descriptions

UserPresenter

UserPresenters purpose is to provide information to be displayed to the LoginView; it also interfaces with the Authenticator to validate the user information submitted from the LoginView.

Name	Description
HelpMessage	Defines the messages to be displayed in the LoginView.
Authenticate	Validates the submitted string from LoginView with the Authenticator.

Table 12: UserPresenter Attributes and Methods

Authenticator

Authenticators purpose is to log in a valid a user. It also keeps track of the current logged in user.

Name	Description
CurrentUser	Reference to the currently logged in user.
validUsers	List of predefined valid users to be compared against when logging in.
Login	Sets the specified user as currently logged in, but only if it is valid.
InitializeUserbase	Populate the list of valid users.

Table 13: Authenticator Attributes and Methods

User

Users purpose is to encapsulate user information; more attributes can be added to this class as needed.

Name	Description
Name	Specifies the name of the user as a string.
Type	Specifies the type of the user, defined by the UserType enum. Currently, a user can either be of type Administrator or Operator.

Table 14: User Attributes and Methods

3.3 Simulation Subsystem

This subsystem handles the actual logic for defining and simulating the vessels.

3.3.1 Detailed Design Diagram

This subsystem consists of the VesselPresenter, the Simulator, the Parser and the Vessel classes as shown in Figure 24.

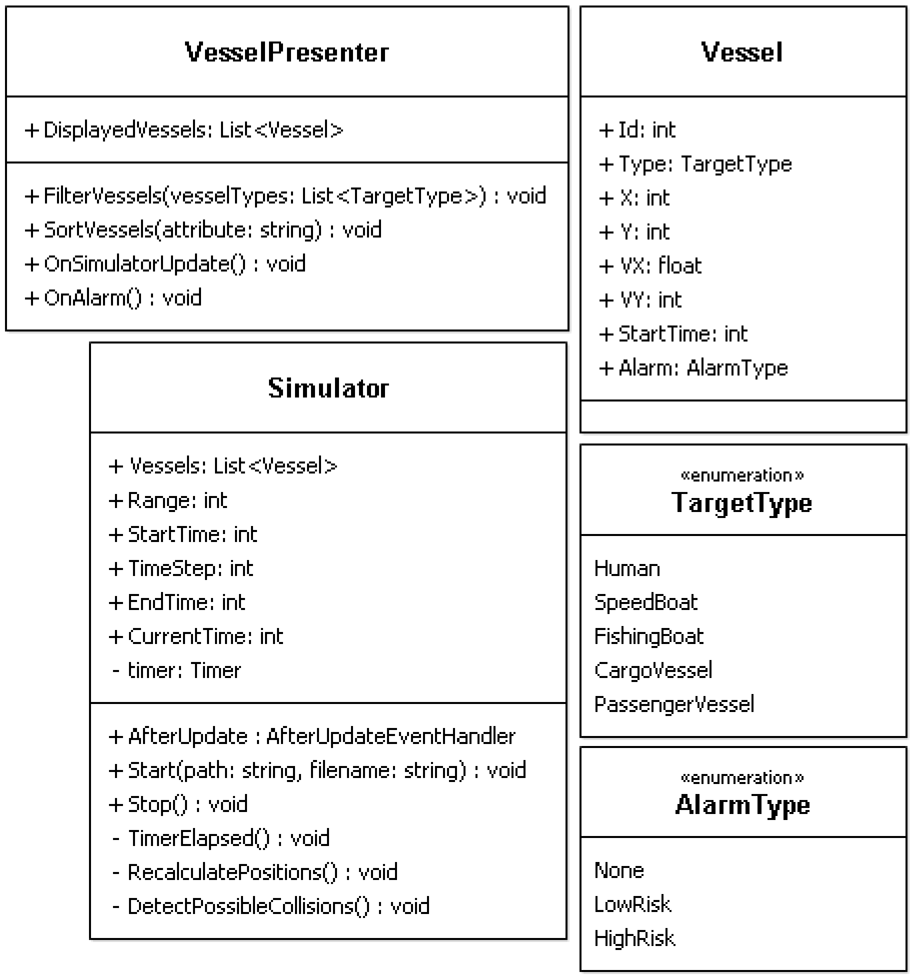


Figure 24: Simulation Subsystem Class diagram

3.3.2 Unit Descriptions

VesselPresenter

VesselPresenters purpose is to provide vessel information to the VesselView, Radar-Widget, and the TableWidget. This includes the filtered and sorted list of vessels, as well as assuring that they are up to date information from the simulator.

Name	Description
DisplayedVessels	List of vessels to be displayed. This is filtered and sorted; it is the final list of vessels to be displayed by the views.
FilterVessels	Filters the vessels according to selected vessel types.
SortVessels	Sorts the vessels according to specified attribute. If the same attribute is being sorted successively, the ordering is toggled every time.
OnSimulationUpdate	Callback event from simulation when vessels are updated.
OnAlarm	Callback event from simulation when an alarm is generated.

Table 15: VesselPresenter Attributes and Methods

Simulator

Simulators purpose is to simulate movement for the vessels according to specified simulation properties. Regular updates are performed in order to achieve constant movement.

Name	Description
Vessels	List of all active vessels to be simulated.
Range StartTime TimeStep EndTime CurrentTime	Defines the properties of the simulation, as specified in the scenario file.
timer	A timer is used to perform updates at regular intervals.
AfterUpdate	Allow events to be registered to execute after every update.
Start	Begins simulation with the specified scenario file.
Stop	Stops the simulation.
TimerElapsed	Function called by the timer at regular intervals. Calls RecalculatePositions and performs the AfterUpdate events.
RecalculatePositions	This is where the vessel positions are updated. DetectPossibleCollisions is then called.
DetectPossibleCollisions	Checks distance between vessels and generate alarms if necessary.

Table 16: Simulator Attributes and Methods

Vessel

Vessels purpose is to encapsulate vessel data.

Name	Description
Id	Id of the vessel as specified in the scenario file.
Type	Type of the vessel as specified in the scenario file. This corresponds to a value of the TargetType enum: Human, SpeedBoat, FishingBoat, CargoVessel, or PassengerVessel.
X Y	Current position coordinates of the vessel.
VX VY	Velocity of the vessel as specified in the scenario file.
StartTime	Time at which to start the simulation, as specified in the scenario file.
Alarm	Specifies the current alarm associated with the vessel. This corresponds to a value from the AlarmType enum: None, LowRisk, or HighRisk.

Table 17: Vessel Attributes and Methods

3.4 Parsing Subsystem

This subsystem takes care of loading initial vessel information from a scenario file.

3.4.1 Detailed Design Diagram

This subsystem consists of the Parser class, but also makes use of the Vessel class as shown in Figure 25.

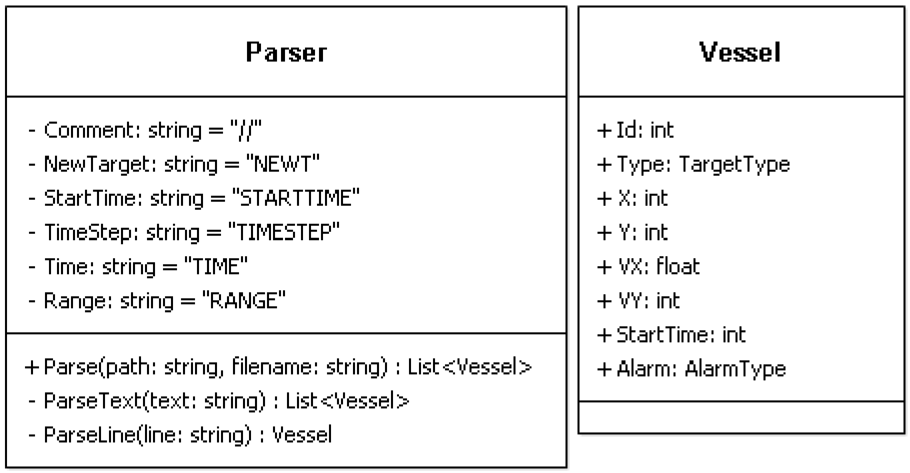

Parser
- Comment: string = "//"
- NewTarget: string = "NEWT"
- StartTime: string = "STARTTIME"
- TimeStep: string = "TIMESTEP"
- Time: string = "TIME"
- Range: string = "RANGE"
+ Parse(path: string, filename: string) : List<Vessel>
- ParseText(text: string) : List<Vessel>
- ParseLine(line: string) : Vessel
Vessel
+ Id: int
+ Type: TargetType
+ X: int
+ Y: int
+ VX: float
+ VY: int
+ StartTime: int
+ Alarm: AlarmType

Figure 25: Parsing Subsystem Class diagram

3.4.2 Unit Descriptions

Parser

Parsers purpose is to load simulation properties and initial vessel information from a scenario file.

Name	Description
Comment NewTarget StartTime TimeStep Time Range	Defines the keywords to look for when parsing the text file.
Parse	Reads the specified scenario file and loads the list of vessels as well as simulation properties. The actual implementation is done in ParseText and ParseLine functions.
ParseText	Reads a block of text and separates it into lines; these lines are then processed using ParseLine function iteratively.
ParseLine	Processes a line of text by determining which keyword matches with the first word. The data is then stored accordingly.

Table 18: Parser Attributes and Methods

4 Dynamic Design Scenarios

Two main dynamic scenarios are filtering vessels and triggering risk alarms. Each is highly dynamic in nature as the states change frequently as the system is running.

4.1 Filtering Vessels

Filtering is activated by the user checking or unchecking a checkbox. As we can see in Figure 26, once a checkbox is modified, the system will remove (or add) the vessels from the list. In Figure 27, the VesselsView asks the VesselsPresenter to return a filtered list of vessel by calling the functio GetVessel() and passing the type of vessel to filter on. The VesselsView then refreshes its list and blips based on the new list.

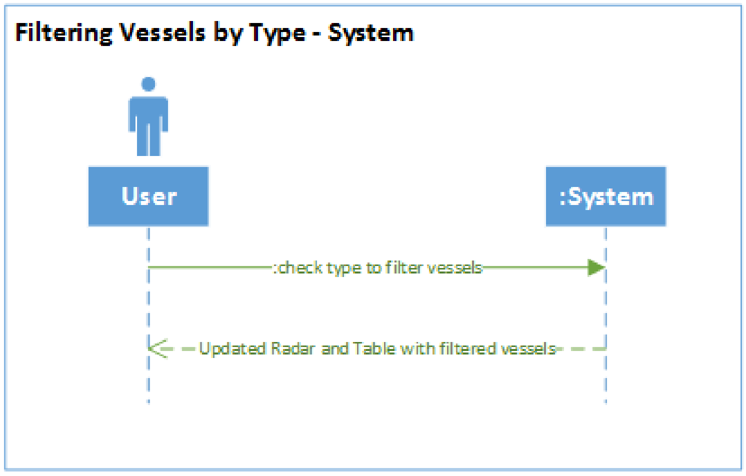

Filtering Vessels by Type - System
User
:System
:check type to filter vessels
Updated Radar and Table with filtered vessels

Figure 26: Filter System Sequence Diagram

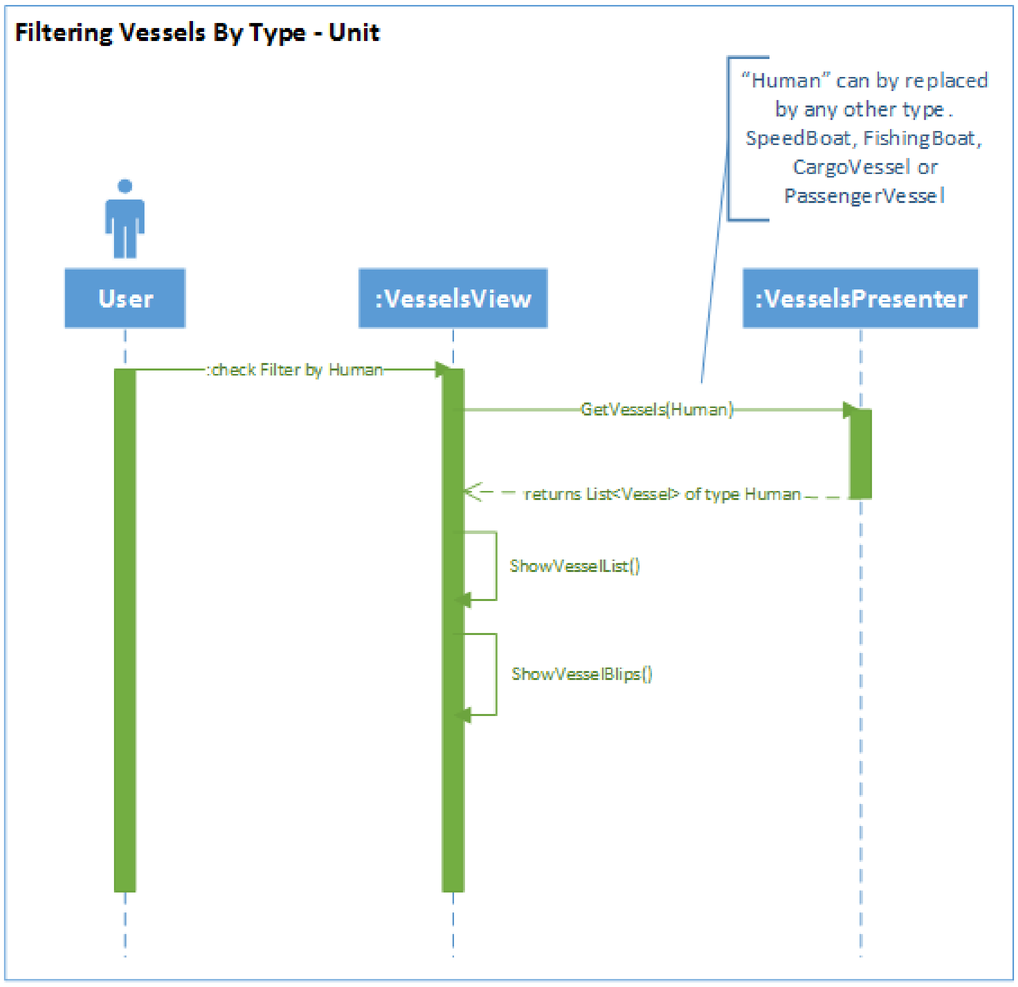

Filtering Vessels By Type - Unit
"Human" can by replaced by any other type. SpeedBoat, FishingBoat, CargoVessel or PassengerVessel
User
:VesselsView
:VesselsPresenter
:check Filter by Human
GetVessels(Human)
returns List<Vessel> of type Human
ShowVesselList()
ShowVesselBlips()

Figure 27: Filter Unit Sequence Diagram

4.1.1 Filtering Contracts

Contract Name	Filter
Use Case	Filtering Vessels by Type
Preconditions	- User must be an Admin. - A checkbox with the filter type must exist in the GUI. - The state of a checkbox on the GUI must be modified.
Postconditions	- The table is refreshed with the filtered list. - The radar is refresehd with the filtered list.

Table 19: Filter Contract

4.2 Triggering Risk Alarms

Triggering risk alarms is activated when at least two vessels are less than 200 meters of each other. As we can see in Figure 28, the VesselsPresenter subscribes to an event that is raised when a possible collision is detected. In Figure 29, the VesselsPresenter first subscribes to the OnAlarm event. The simulator at every time step recalculates the new position of the vessels in vessels list. It calls DetectPossibleCollisions(), and if a possible collision is found it automatically raises the OnAlarm event.

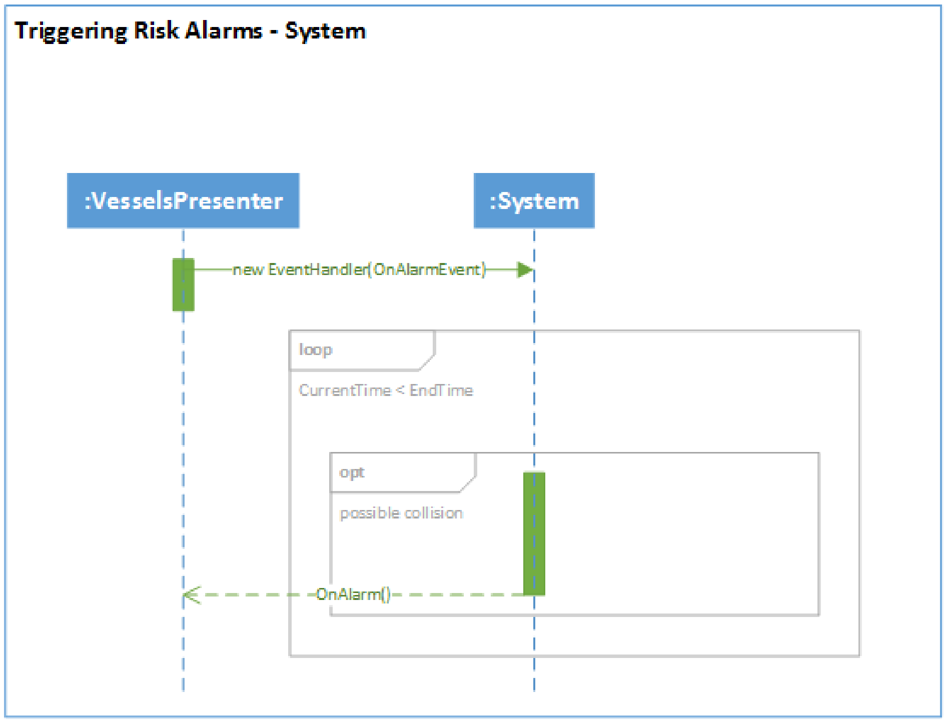


Figure 28: Trigger Risk Alarms System Sequence Diagram

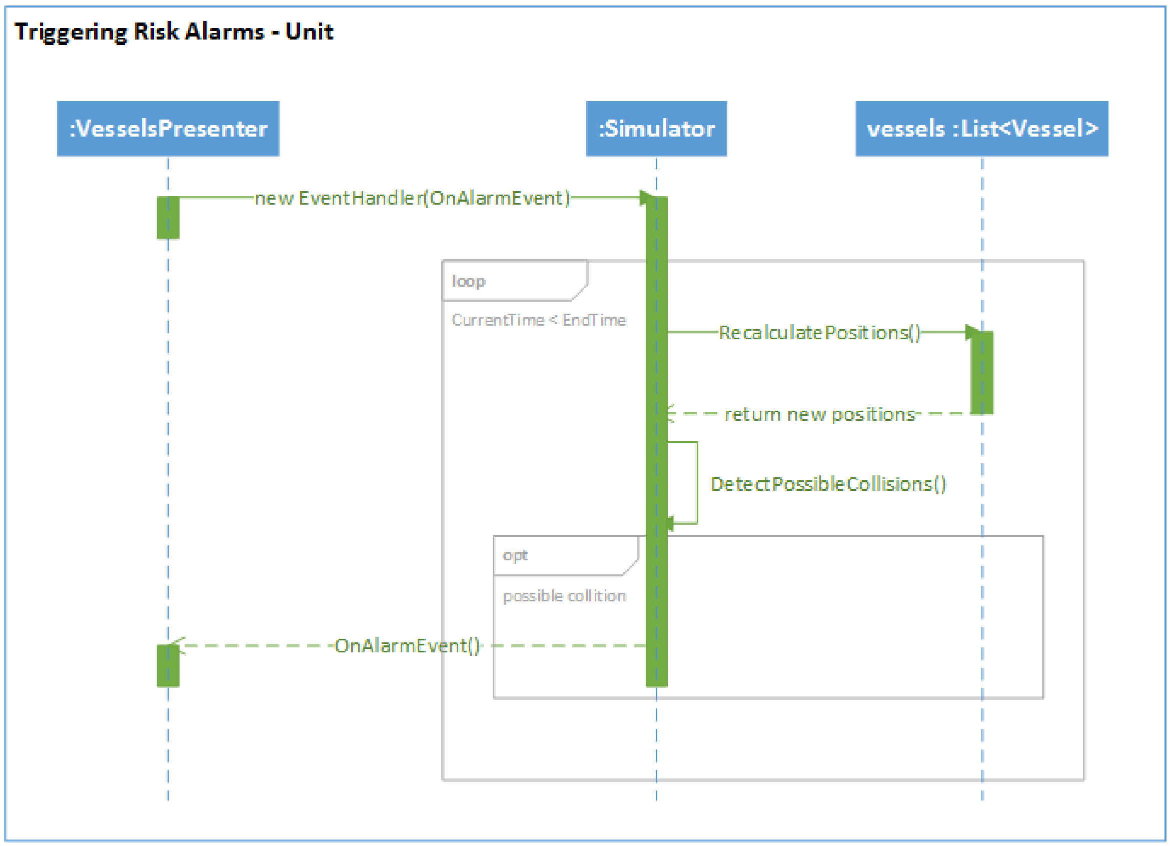


Figure 29: Trigger Risk Alarms Unit Sequence Diagram

4.2.1 Triggering Risk Alarms Contracts

Contract Name	Trigger Risk Alarms
Use Case	Triggering Risk Alarms
Preconditions	- Two vessels must be less than 200 meters of each other together.
Postconditions	- The OnAlarm event is raised.

Table 20: Trigger Risk Alarms Contract

5 Revised Cost Estimation

5.1 Resources Re-evaluation

In line with Delivery 1, some aspect of Resources allocation and planning will be discarded: Physical Resources (e.g.: Machinery, Vehicule, etc), Real Estate Resources (e.g.: Building, Rent, etc) and Financial Resources (e.g.: Financing, Budgeting, etc.) will once again be ignored. Also, Human Resources allocation is directly inherited from Delivery 1 to this document, and will not be covered again. Technical Resource allocation and consumption has not changed as well, therefore it will also be inherited from Delivery 1. One final item will not be modified from Delivery 1 is the Risk assessment. Considering that Risk requires some dedicated resources to handle or prevent it, we will assume the same risk levels and dedicated resources to it.
On the other hand, the scheduling and time allocations have greatly changed from the initial scheduling in Delivery 1 to now, as the requirements for Delivery 2 and 3 have changed, and so did our methods.

5.2 Updated Schedule

In order to be more in line with our hybrid Agile XP/SCRUM model, we have modified our anticipated schedule for Delivery 2 and 3, as demonstrated in the following Gantt Charts in Figure 30, Figure 31, and Figure 32 respectively representing charts for the month of June, July and August.

As reference, we have two Milestones left, both indicated on the chart by a blue mark, representation the due dates for Deliverable 2 (D2) and Deliverable 3 (D3). As mentioned earlier, the development time dedicated to getting the code ready to implement the GUI with the VMS together is evenly distributed between development and debugging/testing.

We allow ourselves a two day debugging buffer prior to the implementation period as well as two days of post-implementation debugging period. The implementation time has been estimated unanimously to 3 days by the programming team, which will then lead us to the actually second delivery.

Considering the modifications to the requirements of Delivery 3 versus what was originally planned, the entire ensuing planning done in Delivery 1 had to be modified. And now that Delivery 3s requirements have been disclosed, we can fully plan our next steps.

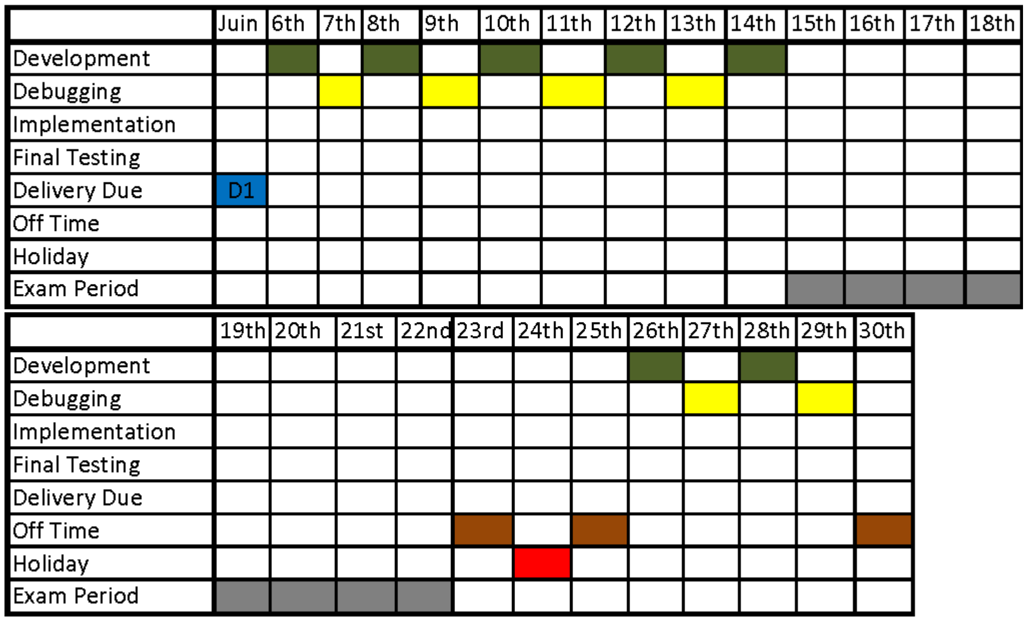
Extract from Delivery 1, regarding the expected requirements of each Deliveries: According to the document Project Information, Section 1, Part 2 (Teams and Roles), each delivery had specific goals to achieve:

- Delivery 1: design of the interface for the core application.
- Delivery 2: creating the design.
- Delivery 3: designing UI

Extract from Delivery 3 requirements:

- Testing & Testing Reports
- Testing Coverage, Items & Cases
- Delivery of the Functioning Systems
- User & Installation Manuals
- Final Cost Revision

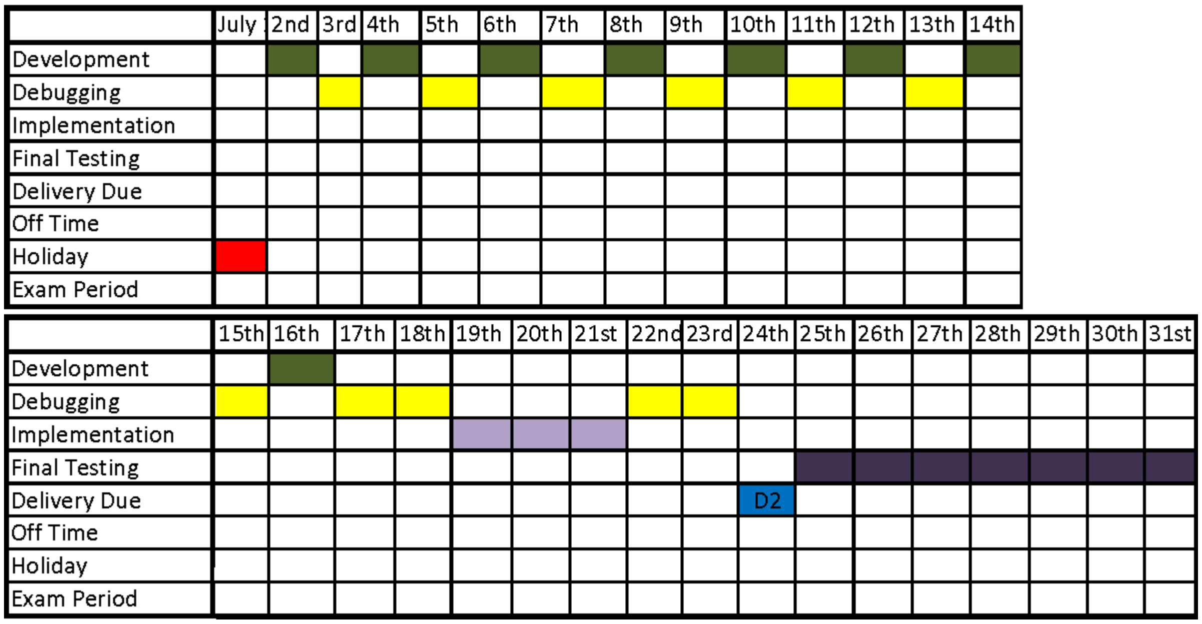
As stipulated across the Gantt charts for the months of July and August, 100% of the time after the handing in of Delivery 2 will be dedicated to the testing phase in accordance with the requirements of Delivery 3. Unspecified, but implied, documentation work will be done on a regular basis both during the completion of Delivery 2 as well as Delivery 3. The redaction of the User and Installation manual will start after Delivery 2 has been submitted. Considering some team members have classes other than this current course, and/or work during the day, we estimate that between 2 to 3 hours will be dedicated daily to the project by team members.



	Juin	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th	16th	17th	18th
Development		🟩		🟩		🟩		🟩		🟩				
Debugging			🟨		🟨		🟨		🟨					
Implementation														
Final Testing														
Delivery Due	D1													
Off Time														
Holiday														
Exam Period											▨	▨	▨	▨

	19th	20th	21st	22nd	23rd	24th	25th	26th	27th	28th	29th	30th
Development								🟩		🟩		
Debugging									🟨		🟨	
Implementation												
Final Testing												
Delivery Due												
Off Time					🟫		🟫					🟫
Holiday						🟥						
Exam Period	▨	▨	▨	▨								

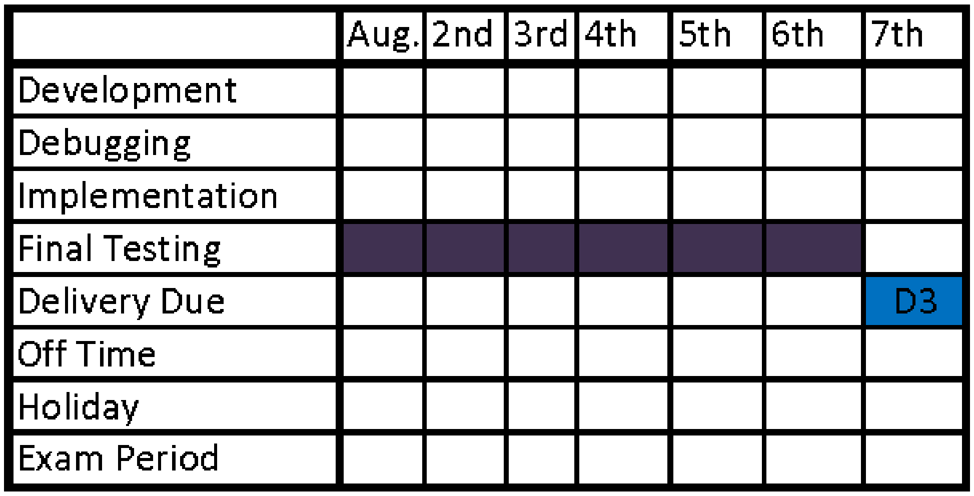
Figure 30: June Gantt Chart



	July	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th
Development		🟩		🟩		🟩		🟩		🟩		🟩		🟩
Debugging			🟨		🟨		🟨		🟨		🟨		🟨	
Implementation														
Final Testing														
Delivery Due														
Off Time														
Holiday	🟥													
Exam Period														

	15th	16th	17th	18th	19th	20th	21st	22nd	23rd	24th	25th	26th	27th	28th	29th	30th	31st
Development		🟩															
Debugging	🟨		🟨	🟨				🟨	🟨								
Implementation					🟪	🟪	🟪										
Final Testing											⬛	⬛	⬛	⬛	⬛	⬛	⬛
Delivery Due										D2							
Off Time																	
Holiday																	
Exam Period																	

Figure 31: July Gantt Chart



	Aug.	2nd	3rd	4th	5th	6th	7th
Development							
Debugging							
Implementation							
Final Testing	■	■	■	■	■	■	
Delivery Due							D3
Off Time							
Holiday							
Exam Period							

Figure 32: August Gantt Chart