MIDDLE EAST TECHNICAL UNIVERSITY

CENG 491

Computer Engineering Design
Detailed Design Report

Perimeter Search and Patrolling Using Limited Capacity Unmanned Air Vehicles

Venture Co.

Efe Suat ERDUR
Basrican ŞEN
Yunus Emre AKBABA
Fırat AKYILDIZ

1. Introduction

This software design document is created by Venture Co. members to be submitted to METU Computer Engineering Department in the concept of design project. The document includes all the design issues for our project. An overview of the problem, data design components and system architecture components are going to be explained in this document.

1.1. Problem Definition

Perimeter Surveillance Systems is a class of radar sensors that monitor activity surrounding or on critical infrastructure areas such as airports, seaports, military installations, national borders, refineries and other critical industry and the like. They mostly use static cameras and sensors. These systems are known as Perimeter Surveillance Radar (PSR). Such radars are characterized by their ability to detect movement at ground level of targets such as an individual walking or crawling towards a facility. Such radars typically have ranges of several hundred meters to over 10 kilometers. These radar systems are quite efficient if the guarded area is well known and it is eligible to construct such platforms, which are exampled above. On the other hand, if the guarded area is unknown and can be changeable, constructing static sensor solutions are useless. Low cost unmanned air vehicles have increasing popularity day by day and can also be used to provide situation awareness in such environments. In the concept of this project, we aimed to solve this surveillance problem in the unknown areas, by creating a multi agent based unmanned air vehicle simulation system.

1.2. Purpose

This Software Design Document provides the design details of perimeter search and patrolling using limited capacity unmanned air vehicles simulation system in the scope of Middle East Technical University Computer Engineering Department Graduation Project.

The purpose of this document is to guarantee that the design will correctly implement all the functionalities identified in the SRS document, it will be easily understandable, efficient, and suitable to change.

1.3. Scope

The scope of this document includes the design patterns of multi agent based unmanned air vehicle simulation system project, brief explanation about the goal, objectives and benefits of the project, constraints, assumptions, dependencies, detailed data description and data dictionary, system architecture with its components, user interface and actions of objects, libraries and tools that are going to be used.

1.4. Overview

This document includes data design and system architecture components of our multi agent based unmanned air vehicle simulation system. To create a perceptible document, the functionalities, behavioral and data models are explained in detailed by using use cases, flow-charts and diagrams. The overview of the document can be described step by step in this way;

In chapter 2, a general description of the software system including its functionality and matters related to the overall system and its design is given.

In chapter 3, assumptions, dependencies and constraints in the design of this system and design goals and guidelines are explained.

In chapter 4, data objects that will be used when implementing the project are explained.

Chapter 5 is related to the system architecture issues.

In chapter 6, user interface is covered detailed. How the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user are explained in this chapter. In addition to this, screen objects and actions are indicated in this chapter.

In chapter 7, libraries and tools that are going to be used when constructing the project are explained.

Chapter 8 is the planning part. It covers time planning of our team.

In chapter 9, there is a conclusion part.

1.5. Definitions, Acronyms and Abbreviations

IDR	:Initial Design Report
SRS	:Software Requirements Specification
DDR	:Detailed Design Report
UAV	:Unmanned Aerial Vehicle
MASON	:Multi-Agent Simulator Of Neighborhoods framework
MAB-UAVSS	:Multi Agent Based Unmanned Air Vehicle Simulation System

1.6. References

[1] MASON: A New Multi-Agent Simulation Toolkit, Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan

[2] OpenMap(tm) : Open Systems Mapping Technology by RAYTHEON BBN Technologies, http://openmap.bbn.com/

[3] JXTA : Juxtapose, http://en.wikipedia.org/wiki/JXTA

2. System Overview

The project is going to consist of building a control and distributed task execution framework to be used with unmanned air vehicles. The goal of our project is basically patrolling an unknown area using limited capacity unmanned air vehicles. Our system will provide a better solution than censor systems when the area to be searched is not eligible to construct censor platforms.

The modules of the project and their missions are explained exhaustive in chapter 4, and Figure 1 shows a general overview of this modules.

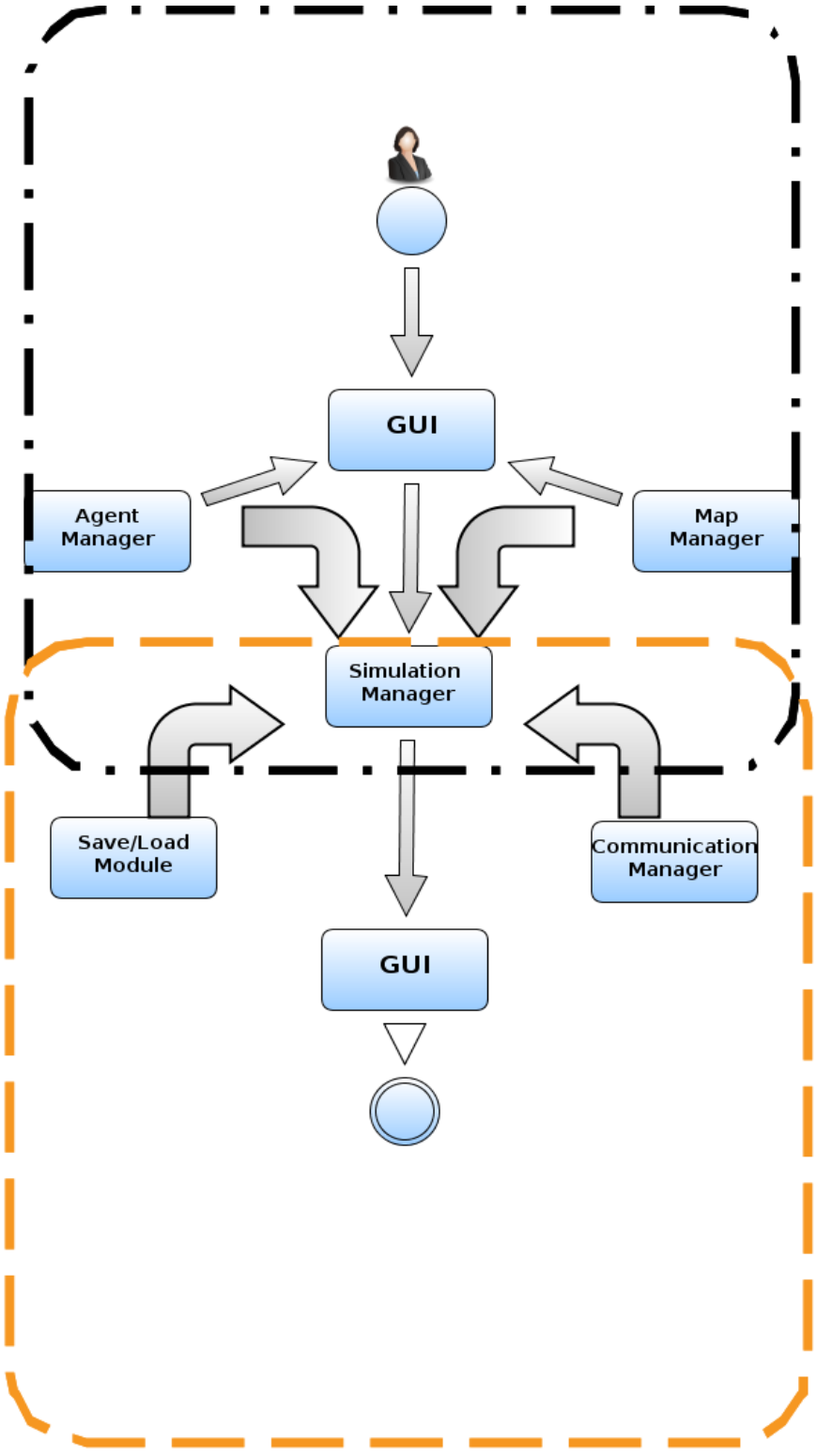

Figure1. Product Overview

3. Design Considerations

In this section, assumptions, dependencies and constraints in the design of this system are described. Moreover, design goals and guidelines are explained in this chapter also.

3.1. Design Assumptions, Dependencies and Constraints

The project will be used by AnelArge to be integrated in real life with real vehicles. It will probably be a part of military service project, so it is not going to be released under open-source free software domain.

The only available language of the system will be English.

In the project, we assume that the weather conditions are stable, which makes us not to include the weather conditions in our modules.

3.2.1 Time Constraints

Since the project is a Computer Engineering graduation project, scheduling and timing are determined and strict. There exist some certain deadlines for the project reports and implementation which is described under chapter 9. The whole project must be implemented, tested and submitted before June, 2012.

3.2.2. Hardware Constraints

For this project, there are not strict requirements about the hardware, but for the users to see the simulation with higher performance without any problems with up to 20 agents, the recommended specifications are listed below;

1. Pentium 4 or more Processor
2. 2-3 GB DDR2 Ram
3. 128 MB Graphics Card
4. 2 GB Hard disk

3.2.3. Software Constraints

1. Simulation can be executed in all of the operating systems.
2. All of the coding will be written with Java Language.
3. Java SDK and JRE

3.2. Design Goals and Guidelines

Our design goals about the project can be listed in this way;
1. Distributed tasks to the separate intelligent agents, which means all of the intelligent agents can make their own decisions to success a given common task
2. Achieving the given task, in this case it is going to be patrolling the whole area
3. P2P communication between the agents which is going to be used in video streaming in the real life
4. A perceptible user interface that gives user the chance to enter some information like map, environment data, number of agents and get feedback from the program
5. Being able to be integrated to the real life with no or little modifications
6. Being able to work on all operating systems

In addition to these, the further specialties that need to be hold are;
Scalability : Only one user can use it on a computer.
Availability : Prototype must be available to its users every time. User can follow the process and be able to interfere at any time s/he wants.
Reliability : Simulation's reliability must be high because after this simulation project company wants to integrate it into a real life project.
Usability : The product can easily be used by any user having knowledge of software programming or not because it will have a simple, user-friendly interface.

4. Data Design

4.1.Data Description

This section describes the data objects that will be used in simulation. At this section we will give information about data classes.

4.1.1. Data Files

All data files will be XML format. We will use SAX parser for the decoding of our XML files.

4.1.1.1. SimulationFile

In simulation file we will have all the changings from beginning of the simulation till the present state of the simulation. The Simulation object will create this data file after all changes in the simulation. The program will delete the SimulationFile after the simulation finishes or user will close the simulation but if the user selects save, it will be saved in SavedSimulations directory under our project file.

Fields	Description
ArrayList<Map <Int , Point>> coordinates	coordinates is a list which contains the all of the agents's id and that their coordinates. We put it in our XML file so that we can load it back.
ArrayList<Map <Int , double>>velocities	velocities is a list which contains the all of the agents's id and their velocities. We put it in our XML file so they can continue the simulation with their saved velocities after loading.
ArrayList<Map <Int , double>>batteries	batteries is a list which contains the all of the agents's id and their battery informations. We put it in our XML file so they will continue with their saved batteries after loading.

ArrayList<Map <Point>> scannedAreas	scannedAreas is a list which contains the coordinates of the areas that is scanned. We put it in our XML file so agents will not scan the same areas again after loading.

Table 1 : Table of SimulationFile

4.1.1.2. ScenarioFile

The ScenarioFile will hold the initial settings of the simulation so that user can use that scenario for other simulations.

Fields	Description
Map map	We will store the map object which user selected in map field so the user can load that map again.
ArrayList<AgentProperties> agentProperties	We will store the list of agentProperties object which user decided their parameters so that UAVs' can be reusable.

Table 2 : Table of ScenarioFile

4.1.1.3. UAVList

The UAVList object will hold the predefined agents. So the user can select one of the real-life agents for the simulation if he/she wants to. The UAVList will be a XML file. We'll use XML parser for get the information of the selected UAV.

Fields	Description
ArrayList <AgentProperties> agentProperties	We will store AgentProperties objects in our file which contains the properties of some real-life agents.

Table 3 : Table of UAVList

4.1.2. Data Classes

We will have 6 modules in our project and that modules will consist this data classes.

4.1.2.1. Task

In task object we will have the list of tasks that core agent created. So that the other agents can reach that tasks and select the most suitable one. In a single task we have the coordinates of the area wanted to be scan and the height values of every coordinate in that area.

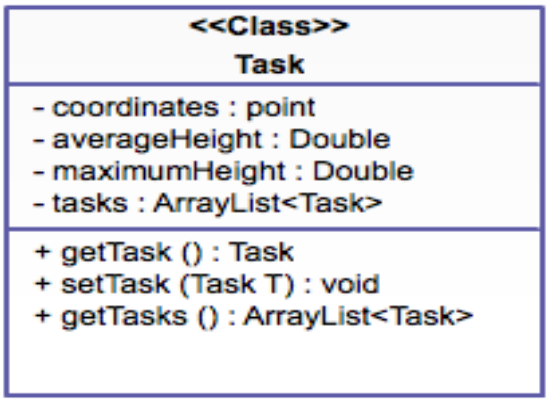

Figure 2 : Class Diagram of Task Object

4.1.2.2. Message

This object will be used for the connections between agents and core. Messages will be send and receive using the module CommunicationManager. There will be five message types in our project.

Message Types

1) When core determine tasks it should send message to all agents “New tasks are ready”.
2) If an agent get a task according to its max height value, it should send a message “Task # is taken by agent #”.
3) If an agent can't complete its task because of its low battery it should send message to core “Task # couldn't being done by agent #”.

4) If an agent completes its task it should send message to core “Task # is completed by agent #”.

5) If all area is scanned core should send message to all agents “All area has been scanned”.

The process of messages is at the beginning of simulation core should create tasks first. After creating tasks it sends message1. So all the agents will take the tasks information. Then they should choose a task from them according to maximum heights they can reach. After taking a task agents should send message2 one by one to core. Now core knows which agent doing which task. Also it has its own job too since its an agent too. While scanning continues if an agent has a problem with its battery it should send message3 to core, share its task's details (percentage of the scanned fields and their coordinates) and leave the field immediately. Core takes that information and when it'll create another tasks it'll use that information so it won't give the scanned fields to another agents as tasks. When an agent finishes its task it sends message4 to core. So core knows that it should create tasks again. It starts to create tasks and share the new tasks to other agents which has finished its task. All scanning progress continues using this protocol. When the area completely scanned core send message5 to all agents so that they understand the work is finished, they can return.

MessageEnum

Value	Name	Description
1	New tasks are ready	Descriptions are in Message Types
2	Task # is taken by agent #	
3	Task # couldn't being done by agent #	
4	Task # is completed by agent #	
5	All area has been scanned	

Table 4 : Table of MessageEnum

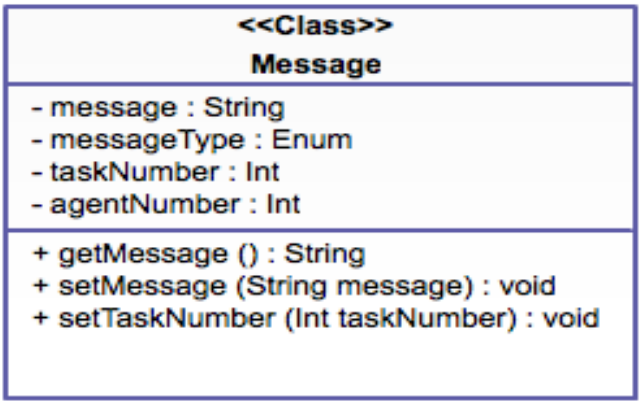

Figure 3 : Class Diagram of Message Object

4.1.2.3. Map

Map object is used for holding the map's attributes because agents will move respect to the map's attributes. Agents will need to know the positions of the current map and the heights of all coordinates of that map. So they will use the values in Map object while they are moving.

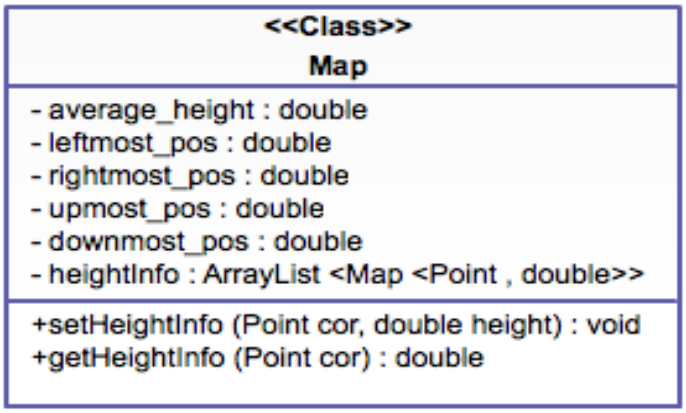

Figure 4 : Class Diagram of Map Object

4.1.2.4. Scenario

Scenario object is used for holding the initial settings of the simulation. It will hold the selected Map Object and a list of AgentProperties Object. If user wants to save that scenario so that he/she can simulate it another time he/she should select save scenario then scenario will be saved in XML format in SavedScenarios directory under our

project file. Also when user will save the simulation the Scenario object should be saved again.

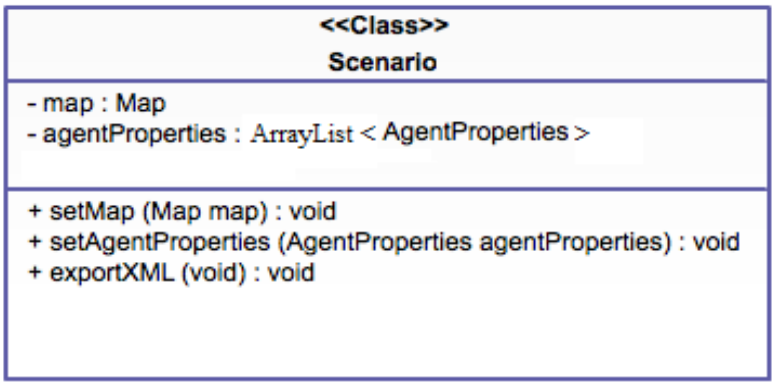

Figure 5 : Class Diagram of Scenario Object

4.1.2.5. Simulation

Simulation object is used for holding the current state of the simulation (agents' coordinates, agents' velocity, agents' battery, scanned area). Also it will write that state to SimulationFile after agents will move to another coordinate. (The map and agents' infos are in ScenarioFile)

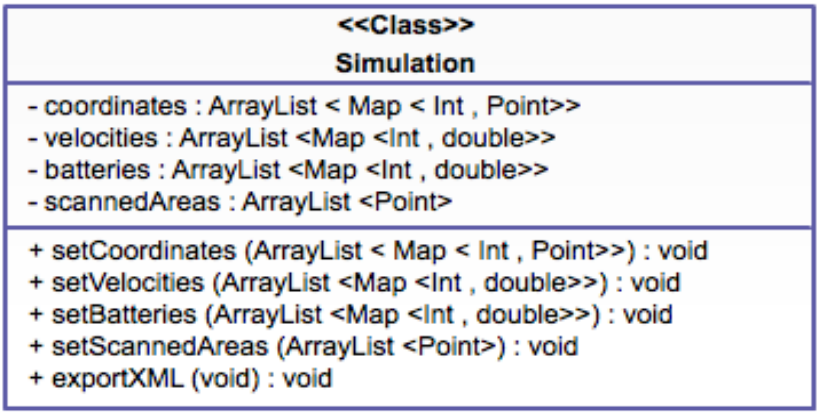

Figure 6 : Class Diagram of Simulation Object

4.1.2.6. AgentProperties

AgentProperties object is used for holding agents' properties. When user wants to add an agent, he/she should set the parameters of it so that the AgentProperties object will hold them.

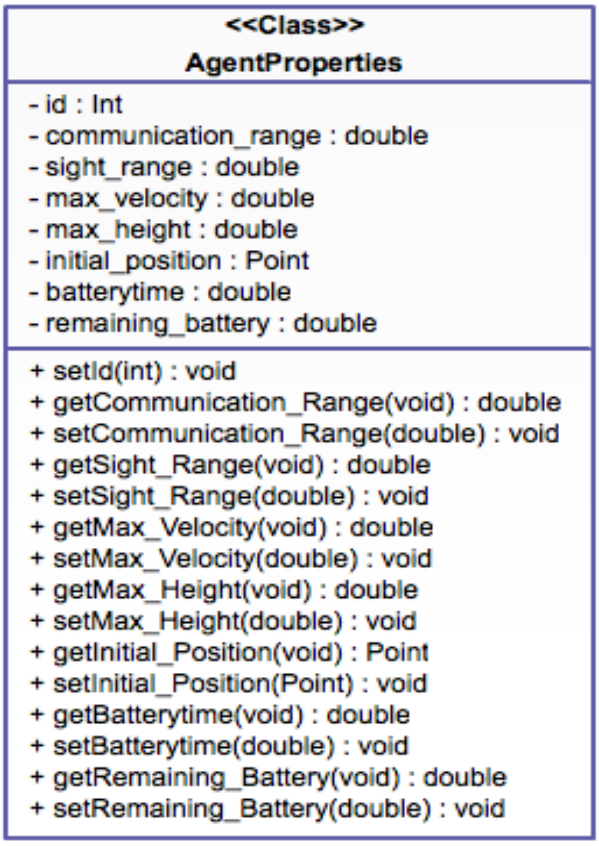

Figure 7 : Class Diagram of AgentProperties Object

4.2.Data Dictionary

Name	Type	Place
AgentManager	Module	5.2.3
AgentProperties	Class	4.1.2.6
CommunicationManager	Module	5.2.5

Interface	Module	5.2.1
Map	Class	4.1.2.3
MapManager	Module	5.2.2
Message	Class	4.1.2.2
Save/Load	Module	5.2.6
Scenario	Class	4.1.2.4
ScenarioFile	XML file	4.1.1.2
Simulation	Class	4.1.2.5
SimulationFile	XML file	4.1.1.1
SimulationManager	Module	5.2.4
Task	Class	4.1.2.1
UAVList	XML file	4.1.1.3

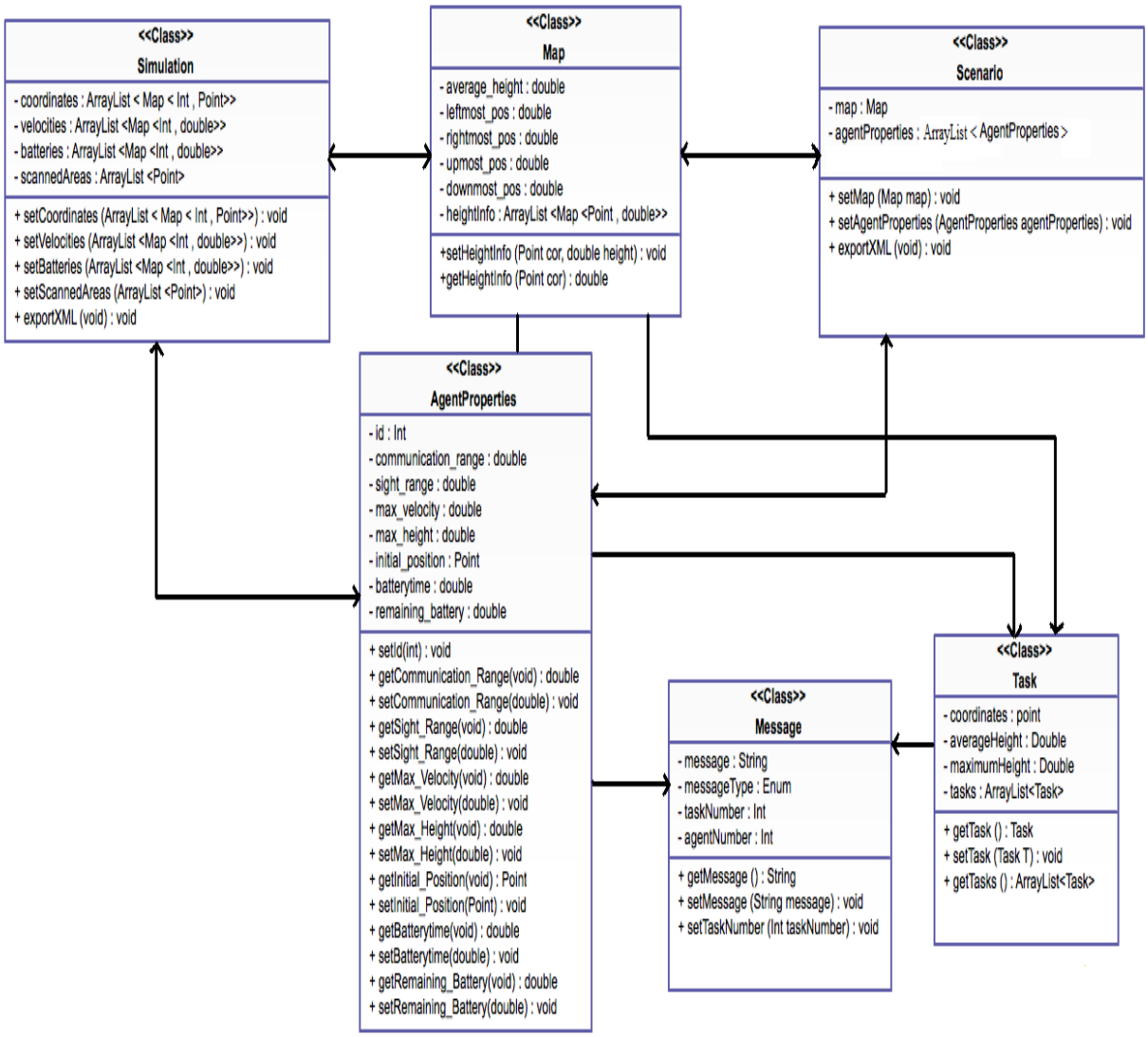

Figure 8 : Data Classes Diagram

5. System Architecture

A description of the program architecture is presented here.

5.1.Architectural Design

In this section, architectural structure of MAB-UAVSS is explained. The MAB-UAVSS consist of 5 main components, which are Interface Module, Communication Manager, Save/Load Module, Simulation Manager, Agent Manager and Map Manager. The architectural diagram is showed below:

Interface Module provides an interface between system and the user. Communication Module handles the internal communication of the agents. Save/Load Module performs all the saving and loading functions that will be supported by MAB-UAVSS. Simulation Manager will act as the core component. It will start ,stop and perform the simulation after all the prerequisites have been met. Agent Manager will be responsible for creating agents and setting required properties of the agents. Map Manager is the component which is responsible for retrieving the geospatial information that will be required for the agents to patrol the area.

5.2.Description of Components

5.2.1. Interface Module

The main purpose of Interface Module is to interact between system and the user. Basically , it is a Graphical User Interface that displays options and menus on the screen for the user to select existing or create new UAV specifications for the agents' properties. It also includes menus for selecting the area that will be scanned.

5.2.1.1.Processing narrative for Interface Module

The functionality of the Interface Module is to construct a graphical user interface between user and system. After user fills the required fields in the GUI, and selects the option to send them to the Simulation, all information is sent to Simulation Manager.

5.2.1.2. Interface Module description

The functionalities and behaviours of Interface Module are widely described in Section 6,with screenshots that were taken from the MAB-UAVSS.

5.2.1.3.Interface Module processing detail

Start
Display the options and menus on the screen
Get input from keyboard or mouse devices
Send input to related components
Display output if necessary
End

5.2.1.4.Dynamic behavior of Interface Module

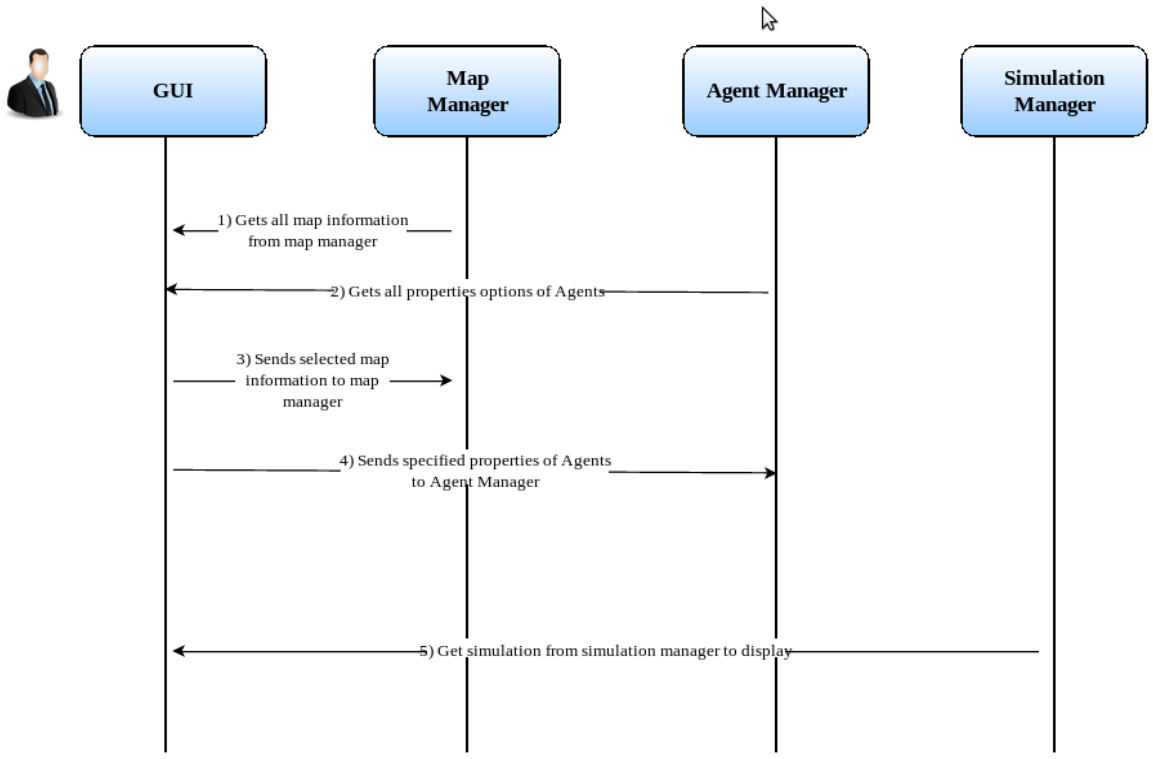

Figure 9: Dynamic behavior of Interface Module

5.2.2. Map Manager

5.2.2.1.Processing narrative for Map Manager

Map Manager component is synchronized with user GUI to supply maps and geospatial information of the maps. The user is going to select a map from GUI and receive map's detailed information.

5.2.2.2.Map Manager interface description

The interface functions of the Map Manager component consist of the functions described in section 4.2. These are the functions which are used to manage scenarios and maps. In conclusion, Map Manager supplies the capability of selecting a map and supplying geospatial information.

5.2.2.3.Map Manager processing detail

Start
Show Maps
Show Maps Detailed Information
Select Map
Change Map
Close

5.2.2.4.Dynamic Behavior of Map Manager

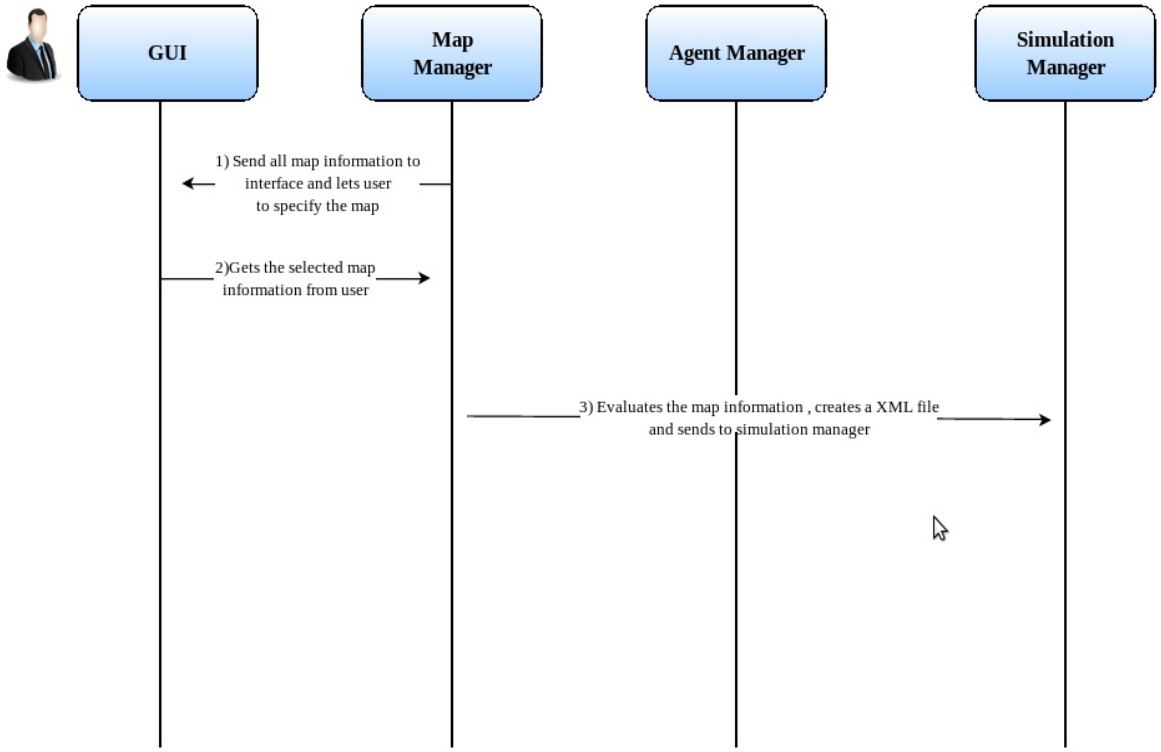

Figure 10: Dynamic behavior of MapManager

5.2.3. Agent Manager

5.2.3.1.Processing narrative for Map Manager

Agent Manager component is synchronized with user GUI and allows the user user to sets the agents' specifications. The user is going to specify the properties of agents from GUI and GUI is going to send the agents' detailed information to agents which is derived from Agent Manager.

5.2.3.2.Agent Manager interface description

The interface functions of the Agent Manager component consist of the functions

described in section 4.2. These are the functions which are used to manage scenarios and agents' specifications. In conclusion, Agent Manager supplies the capability of specifying the properties of agents and improves the reliability of the scenario.

5.2.3.3.Agent Manager processing detail

Start
Specify agents' battery situation
Specify agents' communication range
Specify agents' site range
Specify agents' maximum speed
Specify agents' maximum height
Close

5.2.3.4.Dynamic Behavior Agent Manager

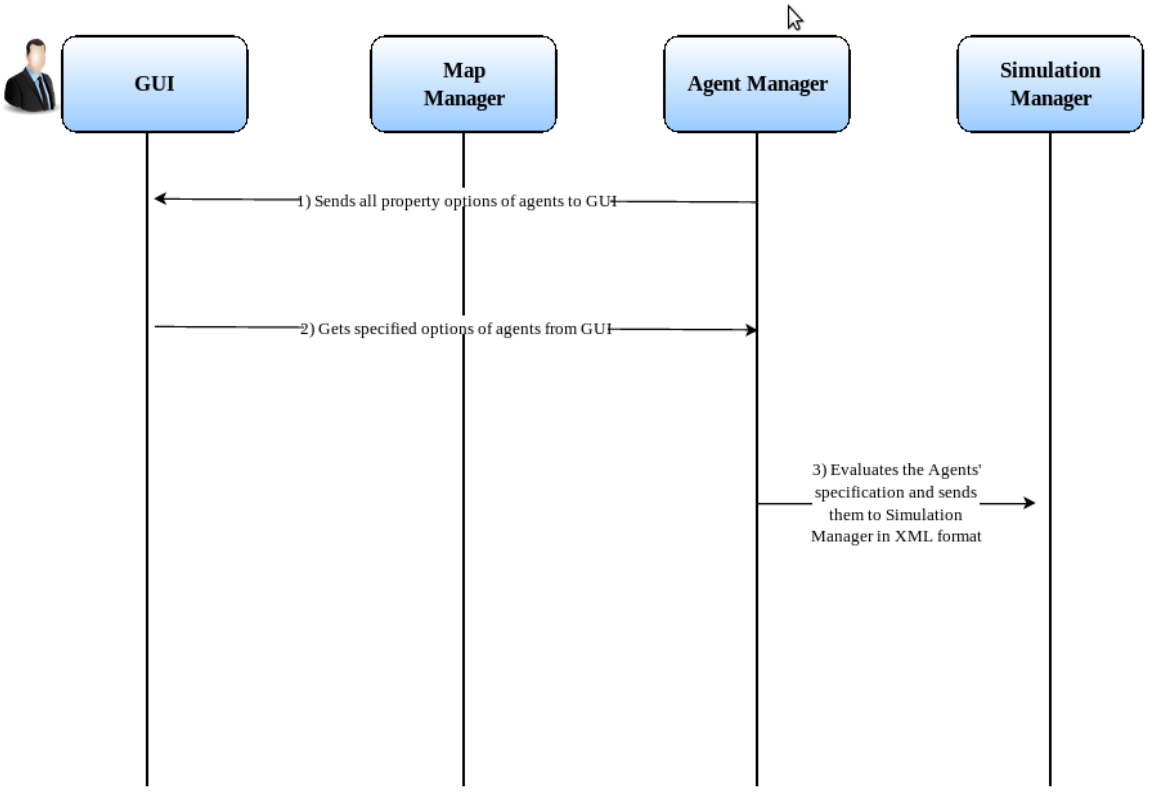

Figure 11: Dynamic behavior of AgentManager

5.2.4. Simulation Manager Component

Simulation Manager, basically, retrieves input from User GUI and Agents, then runs the simulation. Finally, it shows the simulation in user's screen via the Interface Component of the MAB-UAVSS.

5.2.4.1. Processing Narrative for Simulation Manager Component

The main functionality of the Simulator Engine is to simulate the agents using the the coordinates of the agents and the topography.

5.2.4.2. Simulation Manager Component Interface Description

The interface functions of the Simulator Engine component consist of the functions described in section 4.2. These are the functions which are used to manage simulation. In conclusion, Simulator Engine supplies the capability of simulating the scenario and acknowledge the user about patrolling mission.

5.2.4.3. Simulation Manager Component Processing Detail

Start
Wait for a trigger from User GUI
Get required information from Agents
Run simulation
Stop simulation
End

5.2.4.4. Dynamic behavior of Simulation Manager Component

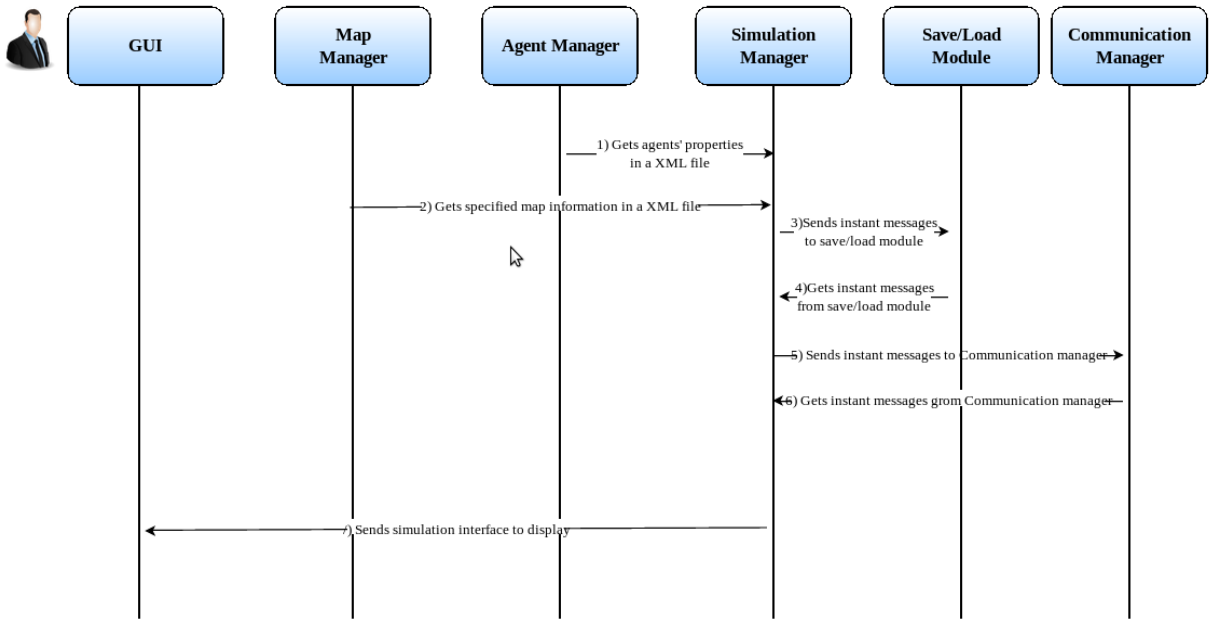

Figure 12: Dynamic behavior of SimulationManager

5.2.5. Communication Manager Component

Communication Manager is the component which is responsible for all the intercommunication of agents in MAB-UAVSS. It will provide functionalities for the agents to receive their tasks from core agent,and send various messages to other agents.

5.2.5.1. Processing Narrative for Communication Manager Component

The main functionality of the Communication Manager is to provide an environment upon which agents can communicate with each other.

5.2.5.2. Communication Manager Component Interface Description

Communication Manager is purely responsible for communicating of agents. Since agents in the MAB-UAVSS are intelligent and make their decisions by themselves, they are not effected for the GUI at the runtime.Therefore, Communication Manager has neither any functions that will be started by user nor any interface.

5.2.5.3. Communication Manager Component Processing Detail

Start
Repeat the 3 steps below until Simulation Manager signals the end of simulation
Wait for a trigger from Agents inside the Simulation Manager
Receive the message or task from corresponding Agent
Send the message or task to the concerned Agent
End

5.2.5.4. Dynamic behavior of Communication Manager Component

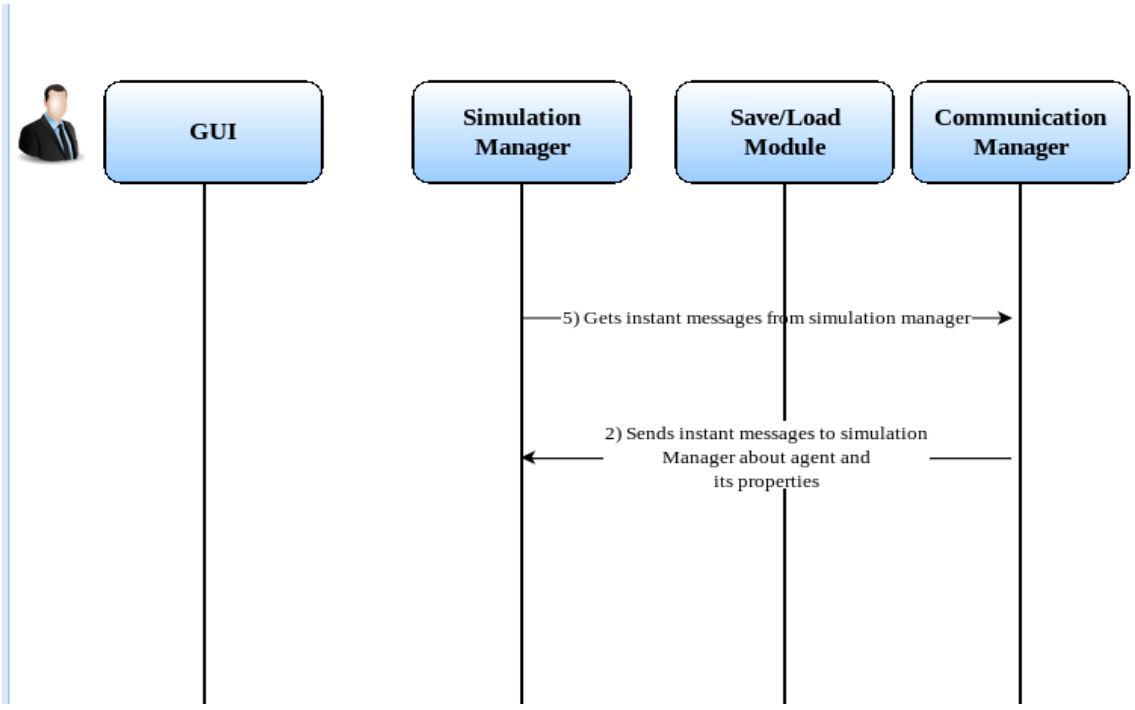

Figure 13: Dynamic behavior of CommunicationManager

5.2.6. Save/Load Module

This component is responsible for all the saving and loading functionalities of MAB-UAVSS.

5.2.6.1. Processing Narrative for Save/Load Module

Save/Load module will perform the requested Save and Load operations.These operations include saving and loading of Scenario, Simulation and UAVList data objects to the disk.

5.2.6.2. Save/Load Module Interface Description

There are options presented in Interface Module for the user to save or load some of the data objects of the system mentioned above to the disk. User can choose the appropriate options from the menus and perform the acknowledged operations.

5.2.6.3. Save/Load Module Processing Detail

Save Function

Start
User chooses to save one of the aforementioned data objects (Scenario, Simulation or UAVList).
Save/Load Module creates an XML file associated with the selected data object.
The newly created XML file is populated with the data which is present at the system.
End

Load Function

Start
User chooses to load one of the aforementioned data objects (Scenario, Simulation or UAVList).

User chooses the file to be loaded.
Save/Load Module checks whether the selected file is a valid one
Save/Load Module reads the contents of the file,filling the data objects in the memory with the data inside the file.
End

5.2.6.4. Dynamic behavior of Save/Load Module

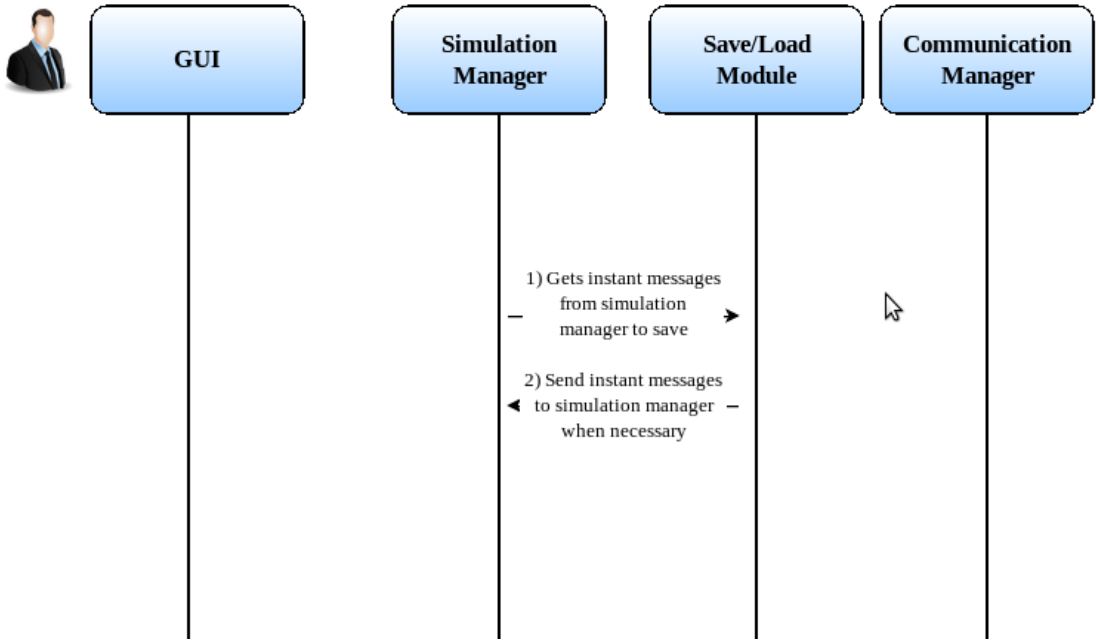

Figure 14: Dynamic behavior of Save/Load Module

5.3.Design Rationale

MAB-UAVSS is multi agent system for unmanned air vehicle to patrol a specified area. We constructed our system with 6 components which are Interface, Agent Manager, Map Manager, Communication Manager, Save/Load Module and Simulation Manager. We at the Venture Co. made our system decomposition as this,because we wanted to follow a strict object oriented approach in our system.So basically, we divided the modules according to the work that they will carry out. In our design, no module should do 2 things that are completely irrelevant, whereas no similar functionalities would be carried out by 2 different components. To make things clear, we will go over

each component one more here.The Interface Module, as its name suggests, will only deal with graphical user interface.Its sole purpose is to take orders from the user and give the produced results back. Agent Manager, on the other hand, is created for constructing and setting the properties of the agents. Map Manager, deals only with geographical aspects of the program. Save/Load Module is only tasked with reading from and writing to files. Likewise, Simulation Manager is only required to execute the simulation.And finally, we have a Communication Manager. This may seem unnecessary at the beginning, since agents themselves may have implemented the communication functionalities among themselves. To clear this point, we may have to state that although our software is intended to be a simulation of unmanned air vehicles(UAV's), the sponsor company plans to implement this project in real life after it has proven efficient in simulation environment. In this real life implementation, agents (UAV's) will communicate with each other using wireless connections whose characteristics are unknown at the current time. So, a communication functionality implemented inside the agents may become an obstacle for the implementing of a new unknown connection method. It may make things harder for our project to go into real life, which is definitely something that we do not want. Therefore, we decided to implement the communication functionalities as a separate module. By this way, sponsor company can adapt the project into the real life by only changing the communication module, without knowing or making changes of the other components, which is one of the main benefits of object oriented programming.

At the start of the project, we decided that one core agent can create separate orders for every single agent and pass the orders to corresponding agents. After a meeting with our sponsor, we have been told that this isn't a multi-agent based approach, and we have decided to change the design. At this stage, we are planning to create intelligent agents which will choose their own orders from a list of available orders created by a core agent.

Another question that appeared is to why the tasks are being in the real time, and not before the simulation begins. As was stated earlier in this document, despite MAB-UAVSS being a simulation product, the sponsor company wants to integrate it into the real life. In the real life, UAV's can be broken or (more likely as in our case), can be shot down. So a plan that has been conducted before the simulation then would be useless. In our system, a plan is created at the real time. So malfunctioning of UAV's during the simulation will not make the intended patrolling operation incomplete. Other agents will take place and patrol the whole area with new tasks created. This is the reason why we are creating our tasks at real time.

6. User Interface Design

6.1.Overview of User Interface

Upon starting the MAB-UAVSS, user will be greeted with a single window.In this screen, user can create a new simulation or load an existing one. After that another window, where the simulation will be presented to the user, will be opened. In addition, after a simulation has been opened, the first window will have functions for monitoring the ongoing simulation. In the next section, the screen images are described with screen objects and actions associated with them.

6.2.Screen Images,Objects and Actions

In this section, some screenshots of MAB-UAVSS are presented with descriptions of them.

The Main Window

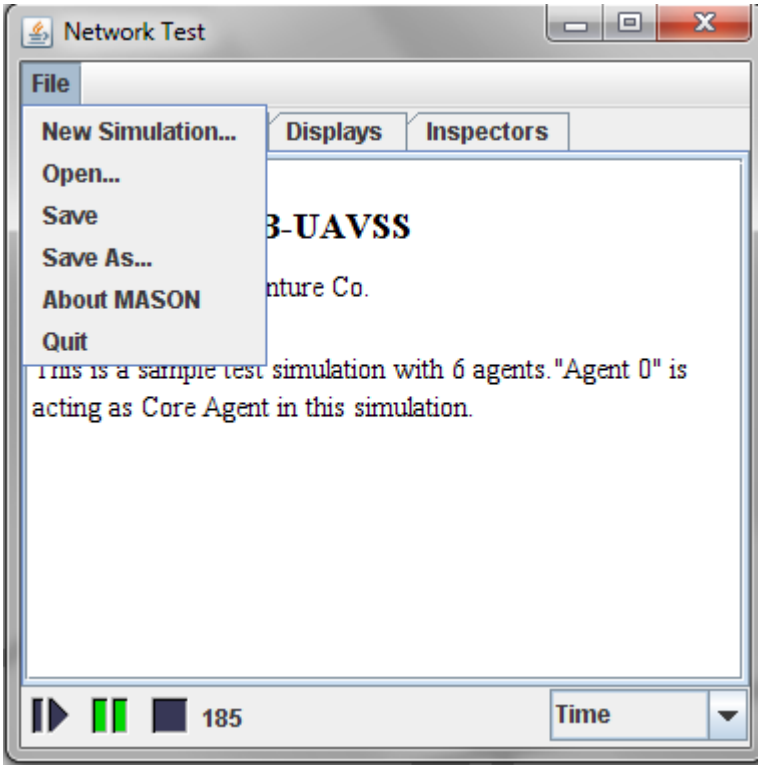

Figure 15 - User Interface Snapshot 1

This small window is opened when MAB-UAVSS is started. In the menu bar, there is only “File” button. As it can be seen clearly in the picture, user can “Create a New Simulation”, ”Load an Existing Simulation” or “Save an Open Simulation”, if only there currently is a simulation running at the given time. The other parts of this screen will be explained later.

The Simulation Window

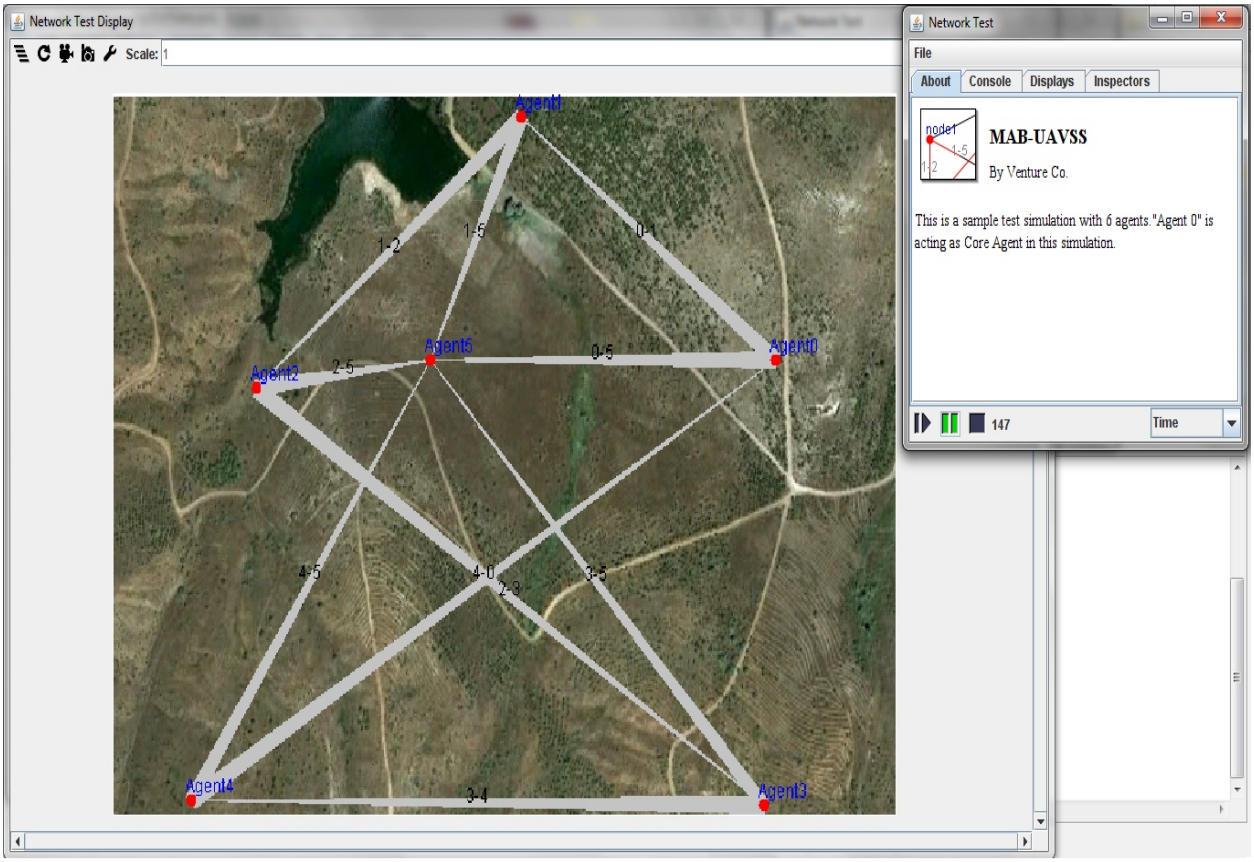

Figure 16 - User Interface Snapshot 2

This screenshot is taken from an ongoing simulation. The big window on the left is simulation window. Agents’ positions on the area is presented, as well as the P2P connections between are shown. The first button to the top left opens a menu in which user can hide and show the connections between the agents, as well as the agents. The second button is for the display frequency. User can choose the interval on which the screen is redisplayed. It can be useful in some scenarios where drawing can take

too much system resources. The third window will allow the user to create a Quicktime movie of the simulation, which requires the Java Media Framework(JMF) to be installed on the computer. Next button is for taking a snapshot of the screen, which would be exported as either in PNG or PDF format, depending of the choice of the user. The last button is for more detailed options about display. To the right of the buttons, there is a textbox named “Scale”. User can zoom in and out by typing values to here. Values greater than 1 will zoom in, whereas values between 0 and 1 will zoom out. Typing negative values in this box is not allowed. After the textbox, there are two more buttons that are not visible in this screenshot. They are for zooming in and zooming out for a fixed amount.In addition, this scaling operation will be done with mouse wheel also.

The Main Window(Continued)
In this part,the rest of the main window buttons will be explained.

The About Tab

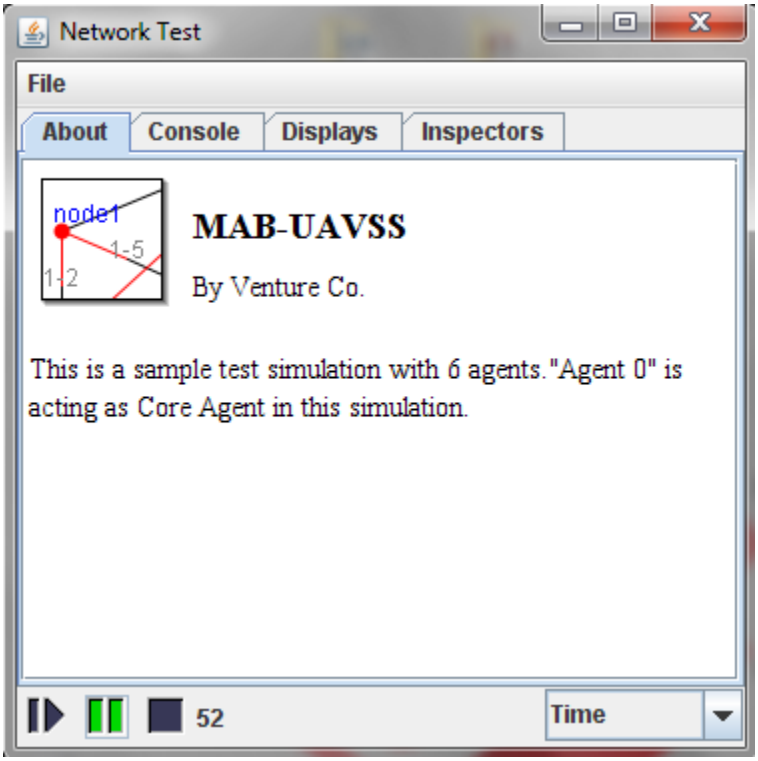

Figure 17 - User Interface Snapshot 3

Some info about the scenario are presented in this tab, as well as a little thumbnail image. To the bottom of the window, there are three buttons. These are Play, Pause and Stop respectively. When the simulation is paused, the Play button is replaced with a “Advance Single Step” button, which will advance the simulation one step. Right of the buttons is the counter for showing how many units has passed since the beginning of

the simulation. It is described as units, because the drop-box to the bottom right of the window determines the unit.It can be “Time”, which will make the units millisecond, or it can be “Steps”, which will make the unit number of steps.

The Console Tab

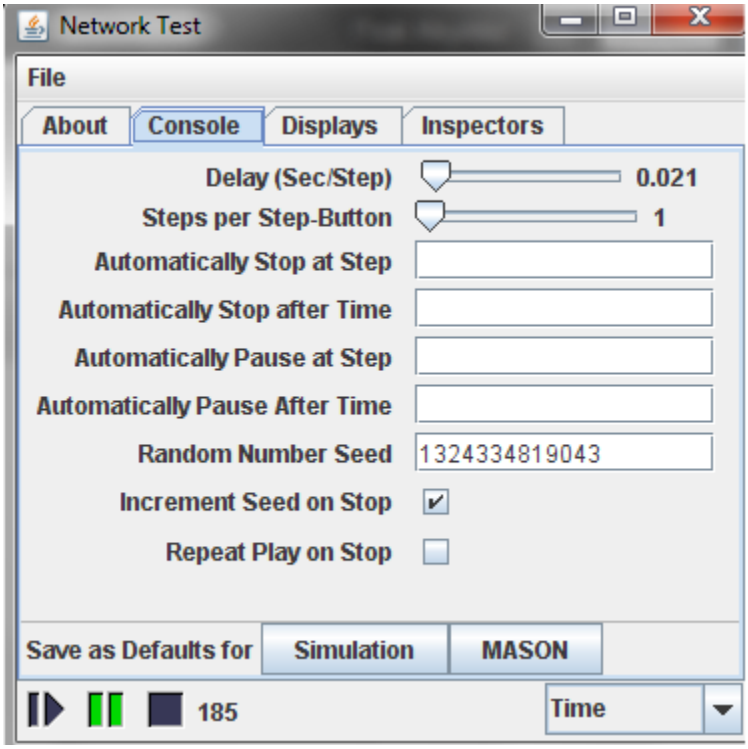

Figure 18 - User Interface Snapshot 4

In this section, user can determine settings about the simulation. The first is the time delay between consecutive steps. The second slider will determine how many steps will be advanced if user presses “Advance One Step” button. The next options are for debugging purposes.

The Display Tab

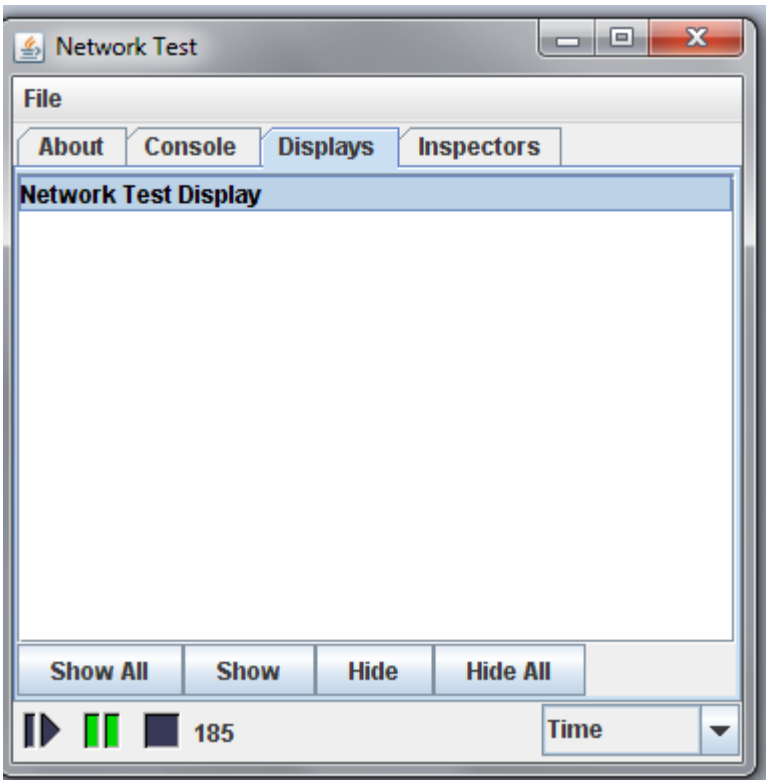

Figure 19 - User Interface Snapshot 5

This tab will show the ongoing simulations. If the user wants to simulate different areas with different agents at the time that another simulation continues, this will be the list of simulations running at the present time.

The Inspectors Tab

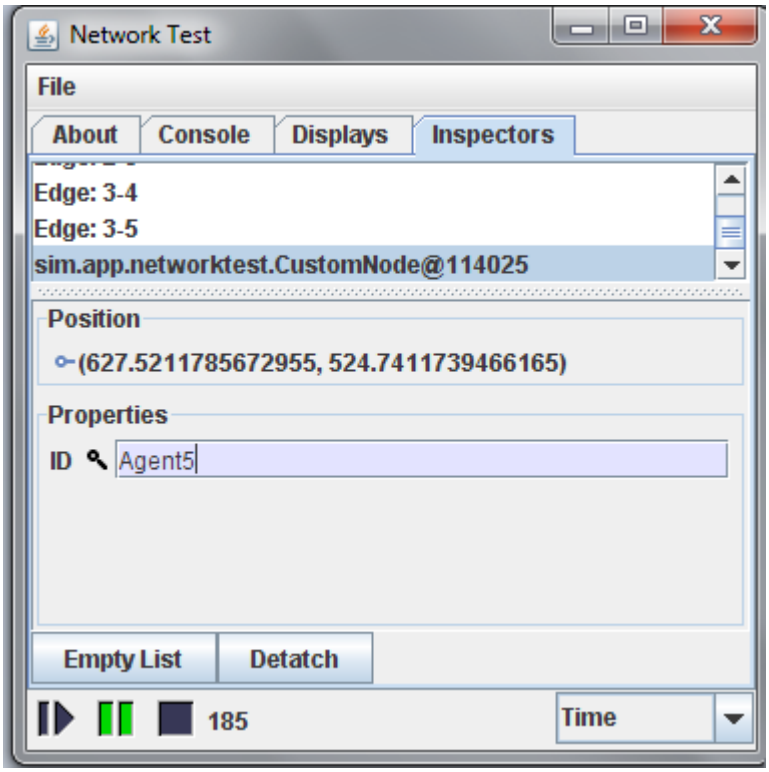

Figure 20 - User Interface Snapshot 6

In this section, user can inspect the elements in the simulation. After user clicks some point in the simulation window, all the elements of the simulation in a little circle is selected and displayed here.

7. Detailed Design

7.1. Detailed Module Description

This chapter contains the internal details of each design component including attribute descriptions for identification, processing and data. In other words, all the details that is needed for implementation are explained in this chapter.

7.1.1 Interface Module

7.1.1.1 Classification

Interface module is a component in this project.

7.1.1.2 Definition

Interface module is a component which is designed for providing the user to interact with the program at any time.

7.1.1.3 Responsibilities

The component is responsible to provide the following functionalities;
-The user can set the parameters for starting the simulation
-The user can choose to create, load and save a simulation
-The user can track the progress of the simulation

7.1.1.4 Constraints

This module needs to be synchronised with SimulationManager module in order to give the user real-time feedback. In addition to this, since the 2D visualization feature of Mason is going to be used, Mason libraries should be imported to the project.

7.1.1.5 Composition

This component does not include any subcomponents.

7.1.1.6 Uses/Interactions

As mentioned above, this module is connected to Simulation Manager during the whole simulation process to give the user real-time feedback about both the simulation

process and the properties of the agents.In addition,it is associated with AgentManager to set the properties of the agents at the beginning of a simulation, and with the MapManager to select an area to be scanned at the beginning of a simulation.

7.1.1.7 Resources

In the visualization part, we are going to use MASON's 2D visualization libraries to inform the user about the simulation process.

7.1.1.8 Processing

This component, in the first place, gets the predefined agents' property parameters from AgentManager, gets the predefined map choices from the MapManager and represents the user some ready choices. Moreover, it provides the user the opportunity to create agents with new parameters according to their wishes. After the user enters his choices (select the map and specify agents' options), the map is sent to MapManager and agents' properties are sent to AgentManager by this module. Figure X shows this steps in a elucidated way;

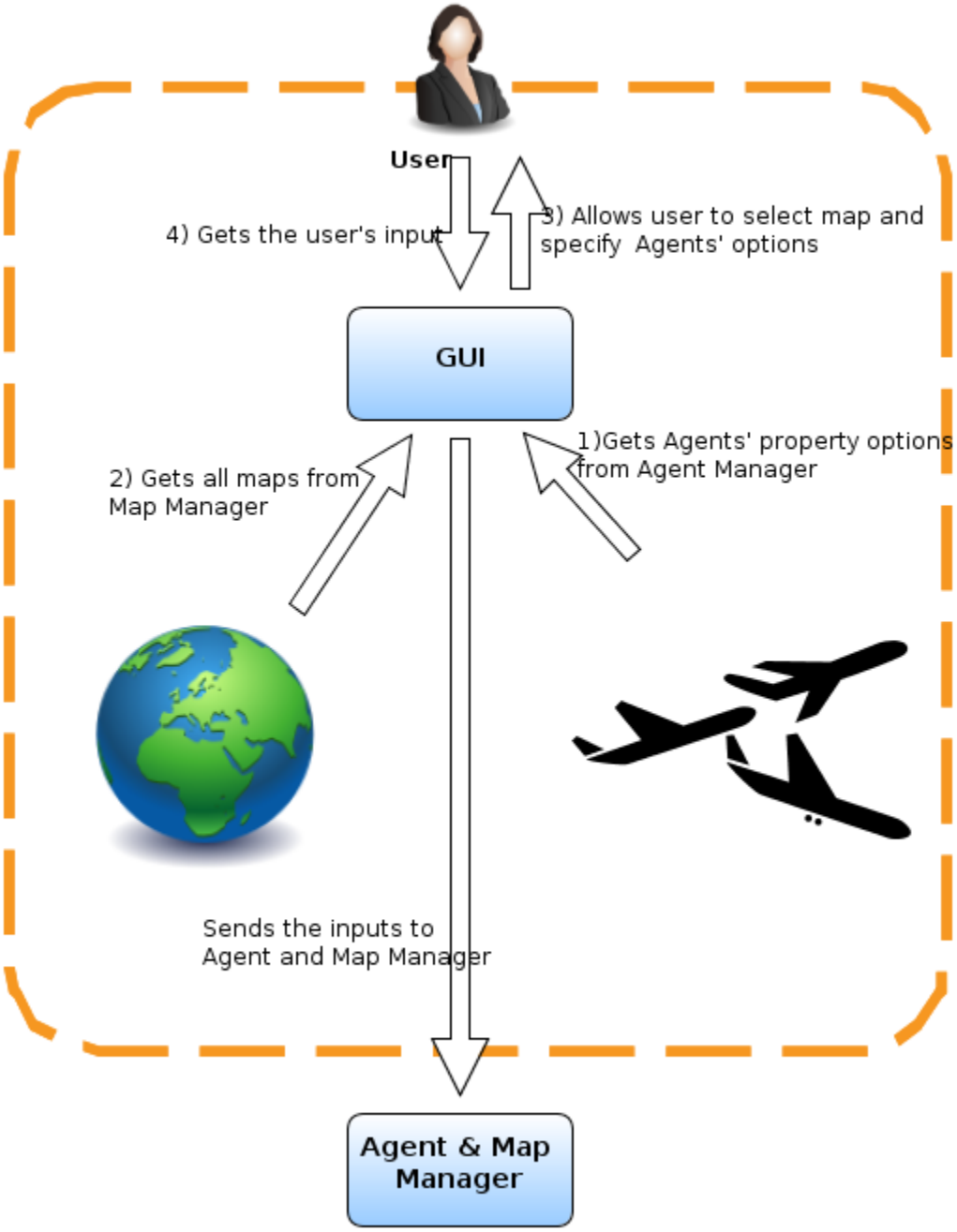

Figure 21: Interactions and Processing schema of Interface

7.1.2 CommunicationManager Module

7.1.2.1 Classification

CommunicationManager module is a component in this project.

7.1.2.2 Definition

The basic purpose of this module is to ensure the communication between agents.

7.1.2.3 Responsibilities

The component is responsible to provide the following functionalities;
- Provide connection between any two agents in the communication range of each others.
- Check the connection between agents if necessary.
- Make the agents to send or receive messages from/to each others.

7.1.2.4 Constraints

Since it is not possible to implement wireless communication (which is planning to be used in the real life) in the simulation environment, in this project peer to peer (P2P) communication environment is going to be used to ta achieve the responsibilities that explained above.

7.1.2.5 Composition

This component does not include any subcomponents.

7.1.2.6 Uses/Interactions

CommunicationManager has only interactions with the SimulationManager.

7.1.2.7 Resources

CommunicationManager will use JXTA protocols for P2P communications between agents of the system. For this protocol to be implemented, we will use JXSE, an open source java implementation of JXTA protocols.

7.1.2.8 Processing

This module gets instant messages form SimulationManager and sends instant messages to SimulationManager along the whole life-cycle of the program. In addition to this it handles communications between agents by P2P protocol.

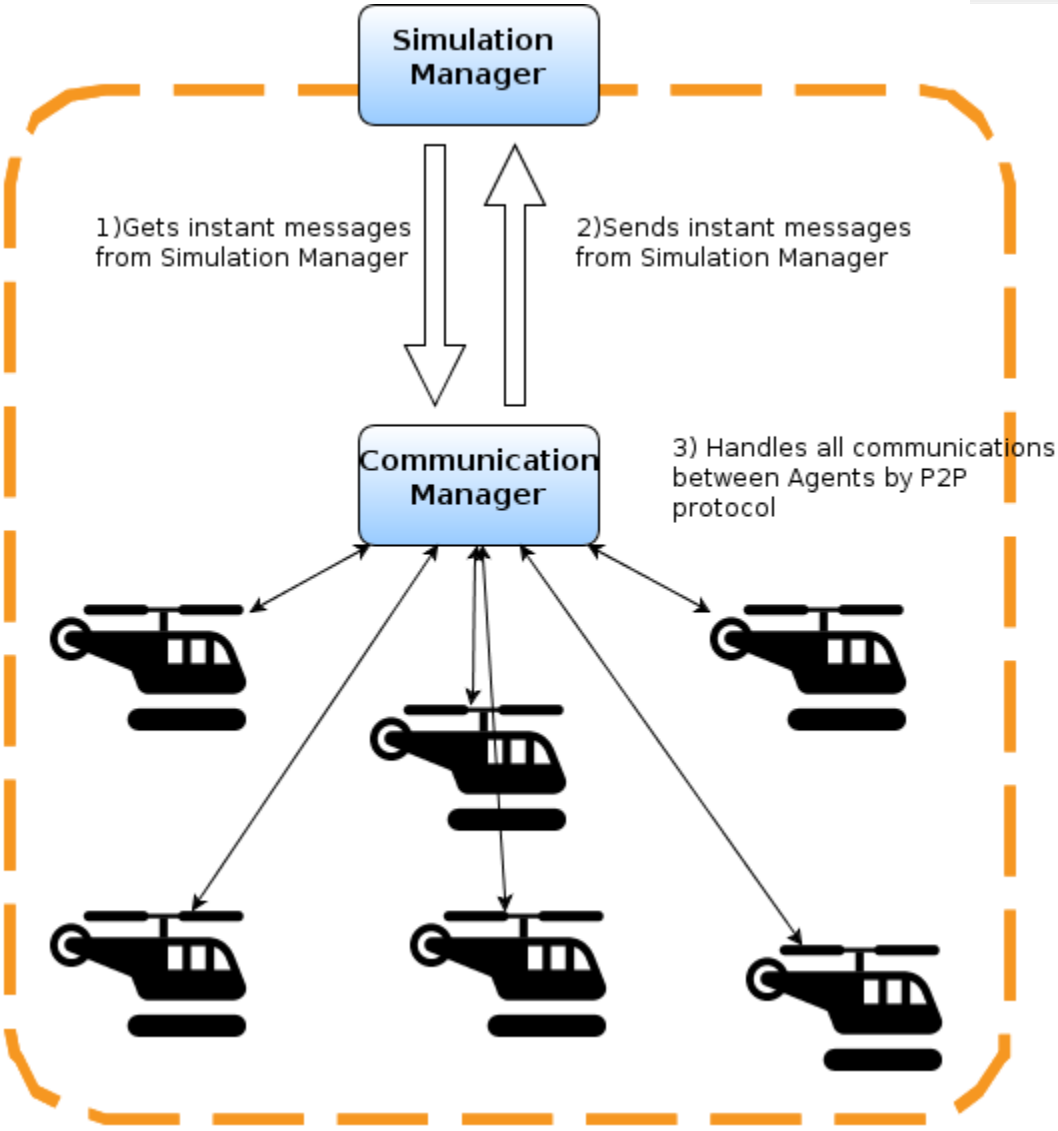

Figure 22: Interactions and Processing schema of CommunicationManager

7.1.3 Save/Load Module

7.1.3.1 Classification

Save/Load module is a component in this project.

7.1.3.2 Definition

This module is created to make the user available to save the processing simulation as an XML file. The user can also load a previously saved simulation and continue processing from where it is saved.

7.1.3.3 Responsibilities

The component is responsible to provide the following functionalities;
- Provide users to be able to save a simulation.
- Provide users to be able to load a previously saved simulation and resume it.

7.1.3.4 Constraints

The save and load operations are going to be performed by using XML files and XML parser tools. For this reason XML parser tools are needed to be imported as a jar file to the project.

7.1.3.5 Composition

This module does not have any subcomponents.

7.1.3.6 Uses/Interactions

This component is directly associated with SimulationManager.

7.1.3.7 Resources

For reading and writing XML's, this component will need an XML parser.

7.1.3.8 Processing

This module gets instant knowledge map, agents and simulation and saves any of these information as XML file when needed. Moreover, it gets necessary XML files whenever SimulationManager needs and sends them to SimulationManager instantly.

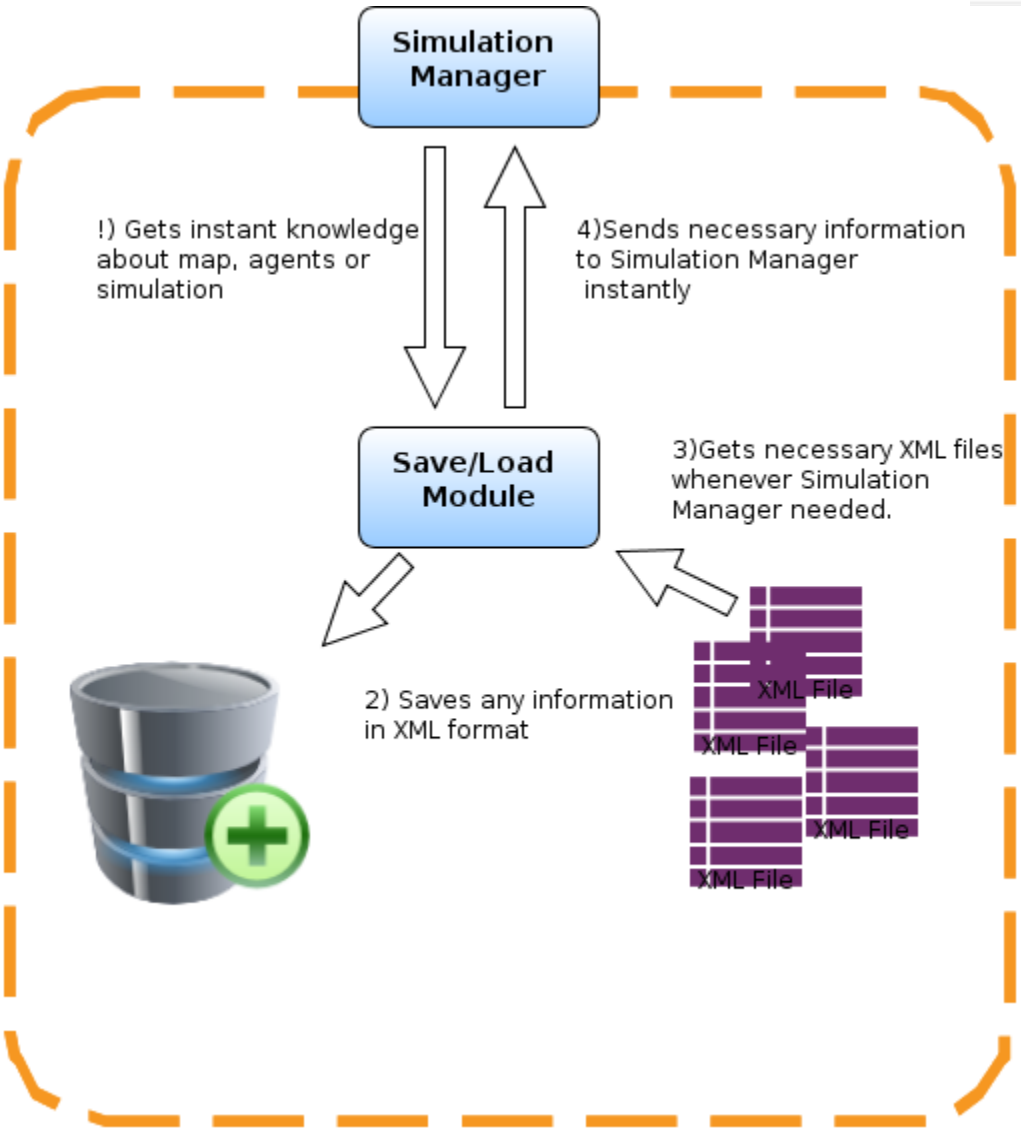

Figure 23: Interactions and Processing schema of Save/Load Module

7.1.4 SimulationManager Module

7.1.4.1 Classification

SimulationManager module is a component in this project.

7.1.4.2 Definition

This module is created to get map and agent information from the interface module and create a simulation.

7.1.4.3 Responsibilities

The component is responsible to provide the following functionalities;

- Creating a new simulation according to the parameters of map and agent objects which are provided by the interface module.
- Creating tasks for agents to perform.
- Ensuring the continuity of the simulation.
- Finishing the simulation when all of the tasks are accomplished.

7.1.4.4 Constraints

No constraints for this module.

7.1.4.5 Composition

This component does not include any subcomponents.

7.1.4.6 Uses/Interactions

Since this module performs the simulation,it is connected to many modules.
First, it uses AgentManager to get agents' properties to run on the simulation. Next, it uses MapManager to get geographical information about the area to be scanned.After that, it will use Save/Load Module for saving and loading simulations and scenarios. In addition, it will need CommunicationManager for the agents to communicate,send and receive orders during the simulations. And finally, it will be connected to the Interface Module for sending real time feedback to the user.

7.1.4.7 Resources

This component will use Mason libraries for performing the simulation.

7.1.4.8 Processing

This module gets selected map information from MapManager and gets agents' properties from AgentManager as separate XML files. After that, it transmits the scenario which is created according to these XML files. It also sends instant messages to Save/Load module and get necessary information when needed. Moreover sending the outputs including current situation of the simulation and position of agents to the Interface module are performed in this module.

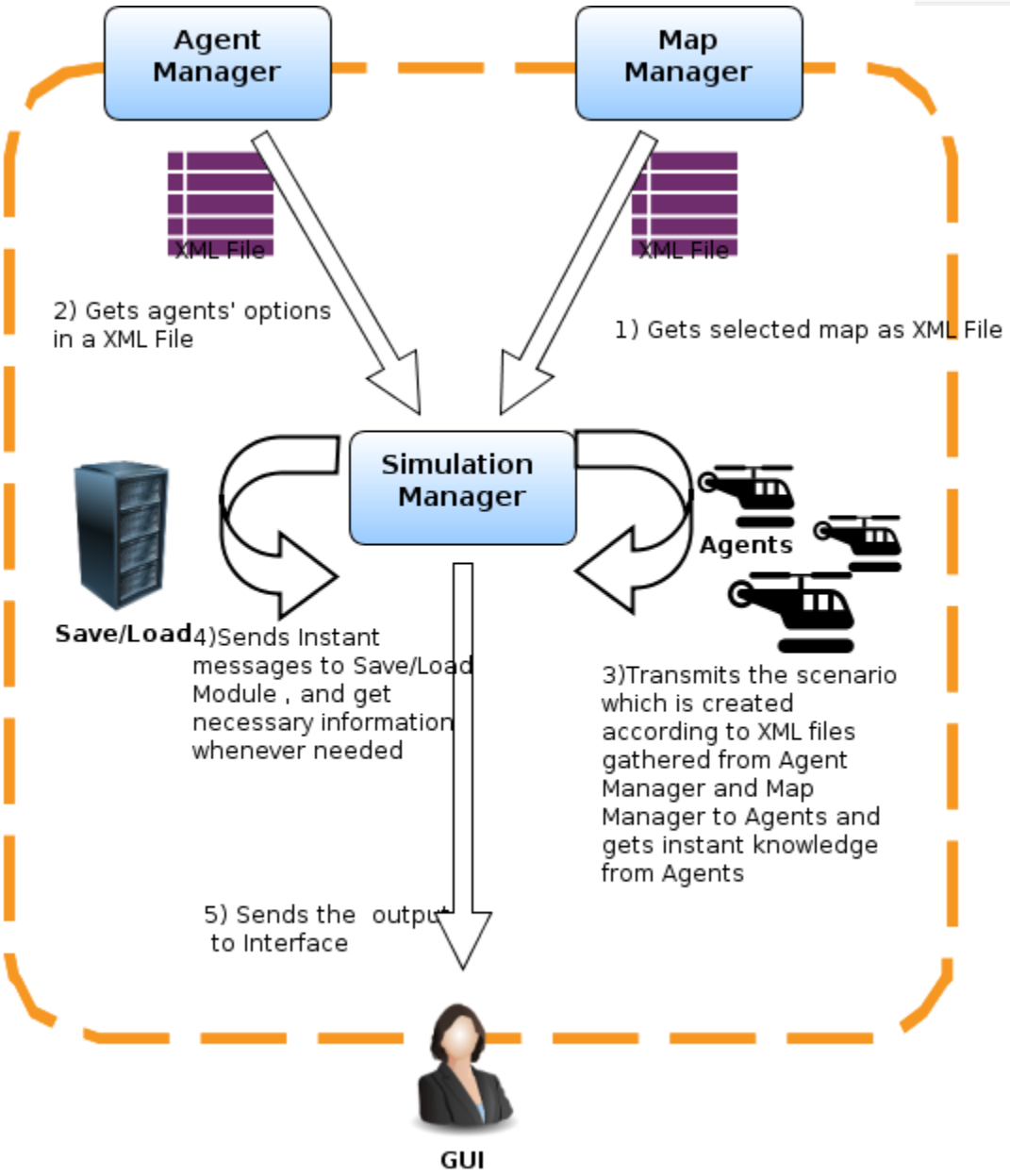

Figure 24: Interactions and Processing schema of SimulationManager

7.1.5 MapManager Module

7.1.5.1 Classification

MapManager module is a component in this project.

7.1.5.2 Definition

The basic purpose of this module is to convert the map that is going to be used to XML format and send it to SimulationManager.

7.1.5.3 Responsibilities

The component is responsible to provide the following functionalities;
- Send interface all of the map choices
- Get the selected map by the user from the interface
- Convert the selected map to XML format
- Send the converted map to SimulationManager module

7.1.5.4 Constraints

Since XML files and XML parser tools are going to be used by this module, XML parser tools are needed to be imported as a jar file to the project.

7.1.5.5 Composition

This module does not have any subcomponents.

7.1.5.6 Uses/Interactions

This module is directly associated for Interface Module for allowing the user to select an area upon which agents will operate. In addition, it is connected to the SimulationManager to send real time geographical properties of the area for agents to work.

7.1.5.7 Resources

This component will use OpenMAP tools for getting realworld geographical data to be used in the simulation.

7.1.5.8 Processing

This module, as the first job, get the maps from OpenMap and send them to Interface for user to select a specific map. After the user select the map, MapManager gets this map information, converts this information to XML file which makes it meaningful to SimulationManager and send this file to SimulationManager.

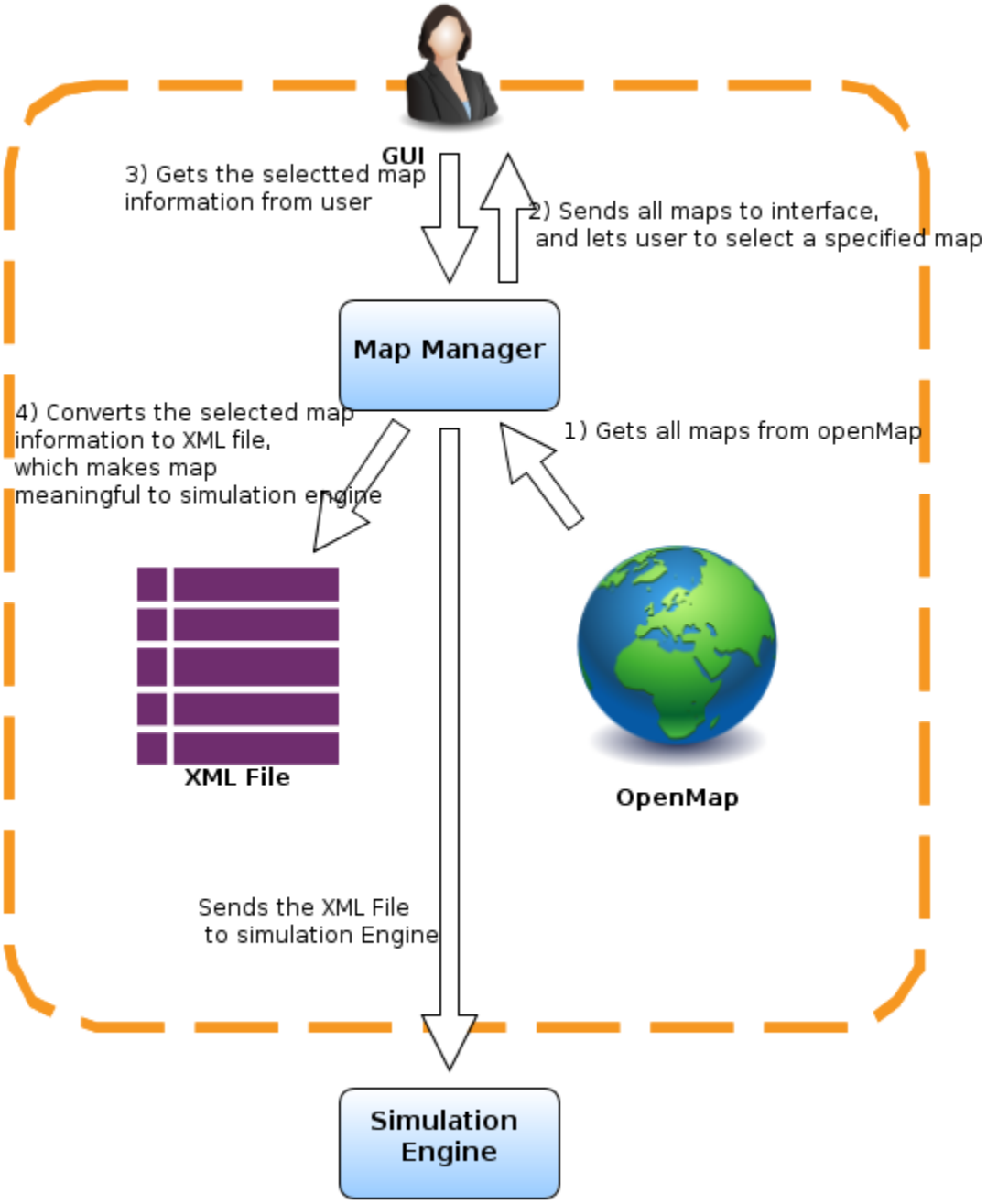

Figure 25: Interactions and Processing schema of MapManager

7.1.6 AgentManager Module

7.1.6.1 Classification

AgentManager module is a component in this project.

7.1.6.2 Definition

The basic purpose of this module is to convert the agent information to XML format and send it to SimulationManager.

7.1.6.3 Responsibilities

The component is responsible to provide the following functionalities;
- Send interface the example agent choices
- Get the selected or specified agent information from the interface
- Convert the agent information to XML format
- Send the converted agent information XML file to SimulationManager module

7.1.6.4 Constraints

Since XML files and XML parser tools are going to be used by this module, XML parser tools are needed to be imported as a jar file to the project.

7.1.6.5 Composition

This module does not have any subcomponents.

7.1.6.6 Uses/Interactions

This components is directly associated with SimulationManager for sending the agent properties and Interface components for getting the properties of the agents.

7.1.6.7 Resources

For reading and writing XML's, this component will need an XML parser.

7.1.6.8 Processing

AgentManager module works in the same manner as MapManager. Gets all predefined agents, represent them to the user by Interface as the first job. After getting the parameters of the agents that user enters from the Interface, it converts this information to XML file and send it to SimulationManager.

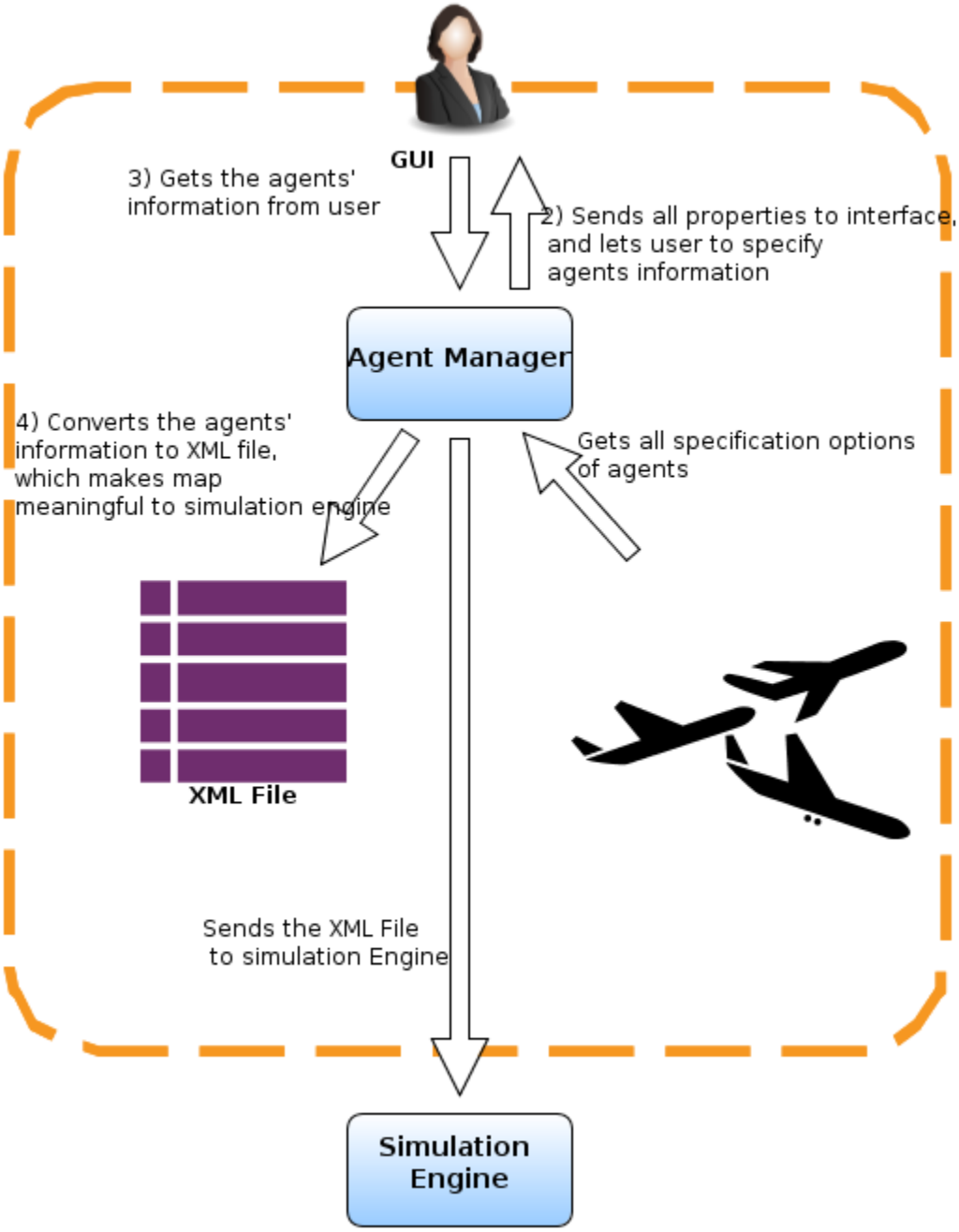

Figure 26: Interactions and Processing schema of AgentManager

8. Libraries and Tools

8.1. MASON

[1] MASON is a fast multi-agent simulation library core in Java, and designed for Java simulations, and also to support much functionality for many simulation problems. MASON contains both a model library and an environment for visualization tools in 2D and 3D. All in all, we are going to use MASON for creating our multi-agent system.

8.2. OpenMap

[2] OpenMap is a free open source programmer's toolkit which is based on JavaBeans. Additionally, OpenMap offers the environment to let users to observe and manipulate geospatial information. Hence, we can get the specified map properties which user selected.

8.3. Eclipse

Eclipse is a free, open-source Integrated Development Environment for software developers. We chose it because we are experienced about using Eclipse and implementing JAVA applications with it.

8.4. JXTA

[3] JXTA is an open-source P2P protocol specification created by Sun Microsystems in 2001. We will use it for peer-to-peer networking between agents. Since JXTA is based on a set of open XML protocols, it can be implemented in Java which is important for us due to compatibility issues.

9. Time Planning (Detailed & Finalized Gannt Chart)

#	Task	Start	End	Duration (days)
	Design Project	10/14/11	6/13/12	173
1	Analysis of the Project	10/14/11	10/18/11	2
2	Pre-Proposal	10/14/11	10/18/11	2
3	Proposal	10/26/11	11/1/11	4
4	Research	10/17/11	11/18/11	24
5	Requirement Analysis	11/15/11	11/25/11	8
6	Framework Research	11/10/11	12/8/11	20
7	Initial Design Report	12/12/11	12/17/11	5
8	Implementation of Agent	12/19/11	1/7/12	15
9	UI Design	12/19/11	1/7/12	15
10	Prototype Demo	1/9/12	1/13/12	4
11	Implementing AgentManager Module	2/7/12	3/13/12	25
12	Implementing MapManager Module	2/17/12	3/2/12	11
13	Implementing Interface Module	3/4/12	3/16/12	10
14	Implementing SimulationManager Module	3/12/12	3/30/12	14
15	Implementing CommunicationManager Module	4/1/12	4/20/12	14
16	Implementing Save/Load Module	4/21/12	5/5/12	10
17	Testing	4/21/12	5/22/12	21
18	User Manual	5/11/12	5/26/12	11
19	Final Demo	5/26/12	6/13/12	12

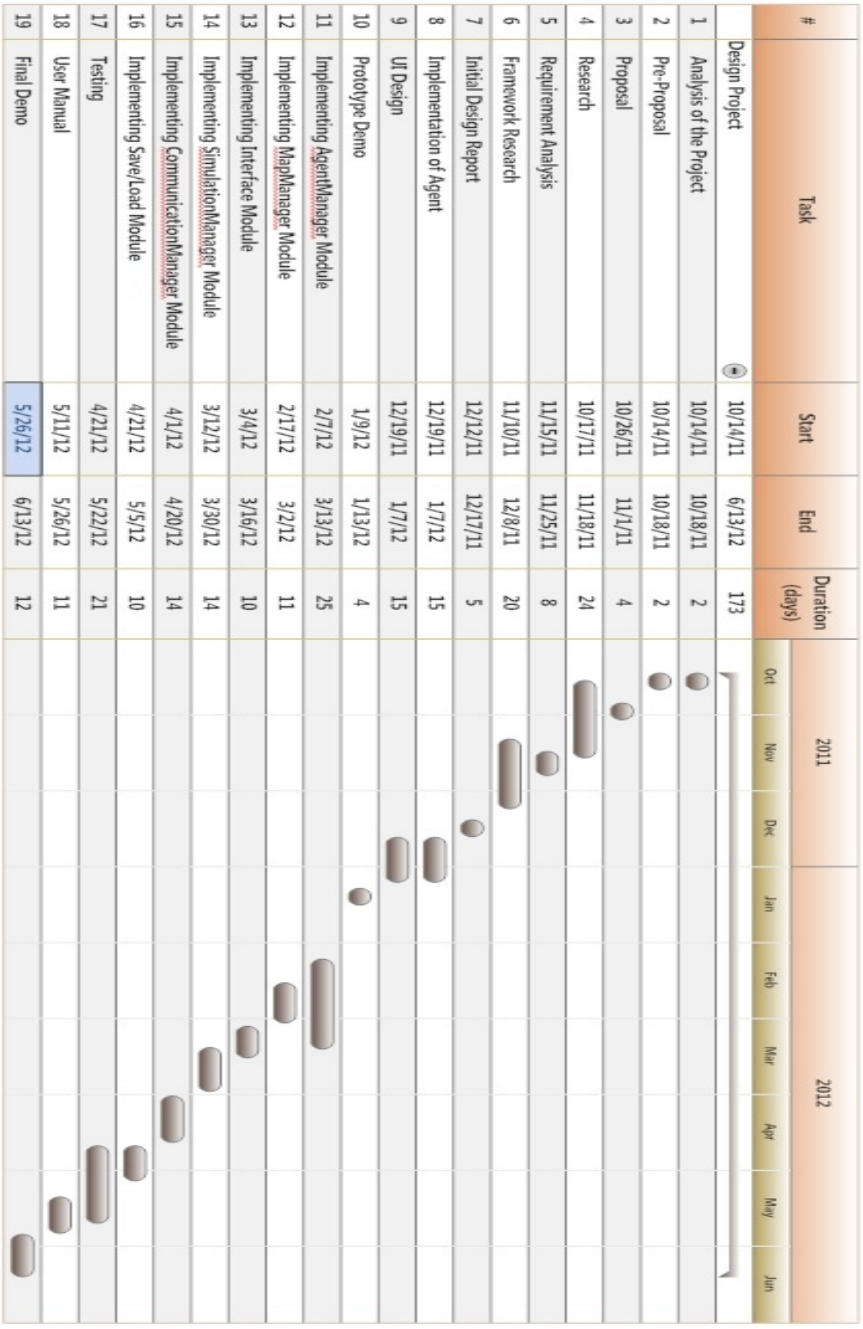

Figure 27 - Gantt Chart

10.Conclusion

This Software Design Document provides a complete description of MAB-UAVSS project supported by Anel ARGE as a final design project of Middle East Technical University Computer Engineering Department, including all design issues. In the first part, the problem is defined, purpose and scope of this document is explained and some definitions and abbreviations that is used in this project are explained. After that, system overview is described briefly and design considerations are explained. Data design part followed these chapters which includes the modules that are going to be used to implement the project. Immediately after that, there exist a data dictionary which explains the functions and function parameters of the modules that were introduced previously. After that, there exists a system architecture part and user interface design part following it. In the end, time planning is described using Gannt charts.

All in all, this documentation aims to describe the design details of perimeter search and patrolling using limited capacity unmanned air vehicles simulation system. It is a bridge between requirements and detailed design, so it will make benefit in the future for understanding and implementing design patterns.