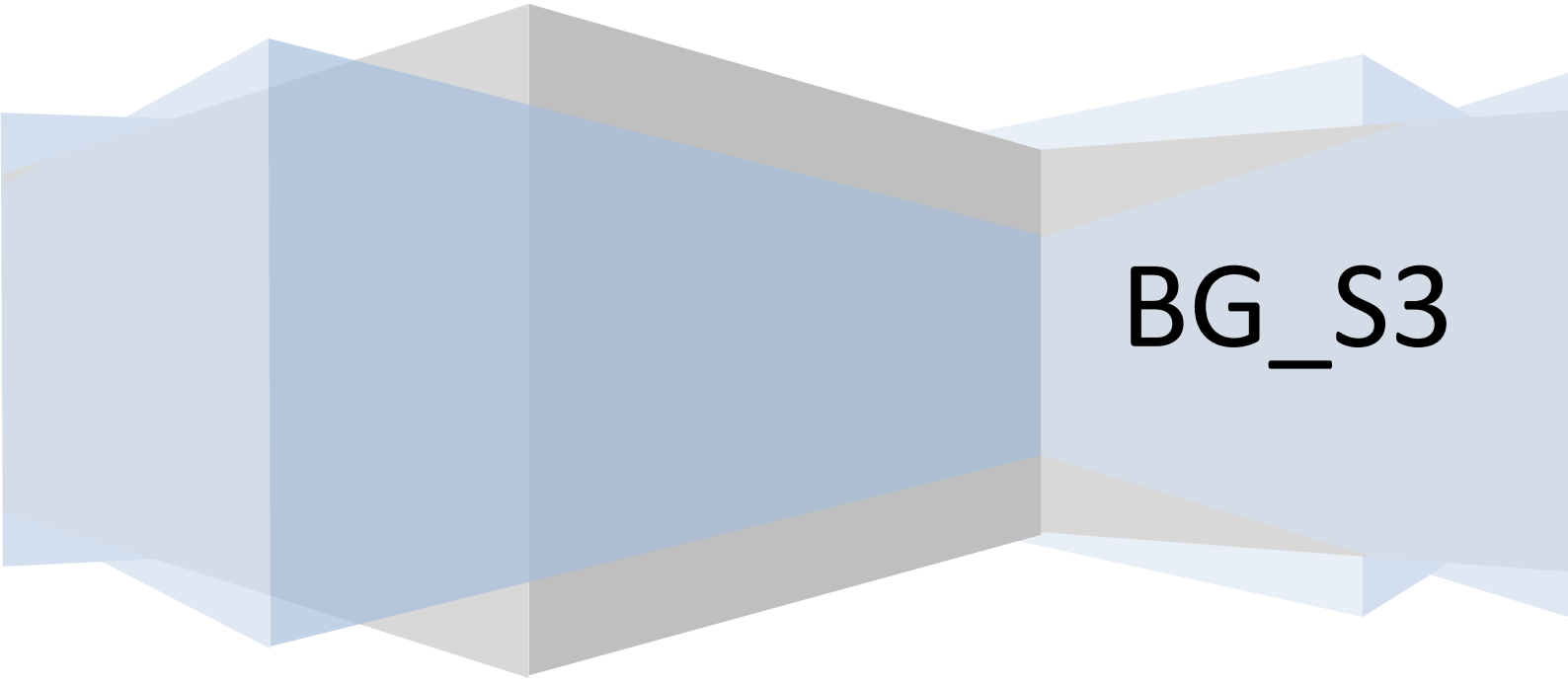
# Detailed Design Report

## Target Tracking by Using Seismic Sensors

Edona Fasllija, Erdoğan Cem Evin, Serkan Yanar, Umut Erdal



# Table of Contents

# 1. Introduction

This is the Detailed Design Report of the Target Tracking by Using Seismic sensors project.

In this document we will give the definition, the purpose and scope of the project. The possible design and other constraints that can be faced will be explained. Data flow models, class diagrams, interface features, entity relationship diagrams, possible use cases will be given within the document.

The document will resemble the last state of the design of the system and will contain all the details needed by the developer to implement the system.

## 1.1. Problem Definition

In this project, the problem that is intended to be solved is motion tracking, i.e. tracking of a moving object within an area where seismic sensors have been previously deployed. The study of this problem is based on signal processing and analysis, which suits our way of solving the problem of the position determination over a field using by means of wireless sensor networks.

## 1.2. Purpose

In this document, we will give the necessary definitions to conceptualize and formalize the design of the software. The aim is to provide a guide to a design that could be easily implemented by any designer reading this report. Many graphical representations and verbal explanations were added to this document to understand each component of the system clearly.

## 1.3. Scope

This report provides the necessary definitions and information about the architecture of Target Tracking by Using Seismic Sensors system. We will give identification, type, purpose, function, subordinates, dependencies, interface, resources, processing and data attributes of each component. We will explain what we have done so far within the schedule and we will mention the future work to be done.

## 1.4. Overview

This is the Detailed Design Report of the Target Tracking by Using Seismic sensors project. In the first section, an overview of the problem and the product is given. Moreover, we will provide some information about terms, acronyms and references which will use on our project. In the second section, functionality, dependencies, limitations, objectives and benefits of our project will be explained.

In the third section, we will give information about the design assumptions, dependencies and constraints. Goals, guidelines and priorities for design of the system's software will also be described.

In the fourth section, we will describe how the major data or system entities are stored, processed and organized. Moreover, we will provide an alphabetical list of the system entities.

In the fifth section, we will give a pictorial representation (UML component diagram), to show the major subsystems, data repositories and their interconnections. Also, the relationships between the components to achieve the complete functionality of the system are explained. Furthermore, there will be a decomposition of the subsystems in the architectural design.

In the sixth section, user interface together with the realizations of the functionality provided to the user are described.

In the seventh section, we will provide the internal details of each design component. We will explain the attribute descriptions for identification, processing and data. We will try to give all the details that will be needed so that a programmer can get enough information about implementation.

In the eighth section, we will give a tentative time plan which shows when we will finish the remaining parts of our project.

The ninth section will give information about the libraries and tools which we are using.

In last section, we will sum up the content of the Detailed Design Report of the Target Tracking by Using Seismic Sensors system.

## 1.5 Definitions and Abbreviations

IDE : Integrated Development Environment

PDF : Portable Document Format

RCP : Rich Client Platform

UML : Unified Modeling Language

GIS : Geographic Information System

GPS : Global Positioning System

ORDBMS: Object-Relational Database Managment System

UI : User Interface

GUI : Graphical User Interface

GPS : Global Positioning System

Jinput : Java library used to process Joystick input

SRS : Software Requirement Specification

IDR : Initial Design Report

DDR : Detailed Design Report

## 1.6. References

Libelium Comunicaciones Distribuidas S.L.. (2010). *Hardware description.* Available: http://www.libelium.com/products/waspmote . Last accessed 9th January 2011.

The Eclipse Foundation. (2012). *Rich Client Platform.* Available: http://wiki.eclipse.org/index.php/Rich_Client_Platform . Last accessed 9th January 2011.

PostgreSQL Global Development Group . (2011). *PostgreSQL*. Available: http://www.postgresql.org/ . Last accessed 9th January 2011.

Geophone specification provided at http://www.oyogeospace.com/technologies.htm.

"Geographic Information Systems as an Integrating Technology: Context, Concepts, and Definitions". ESRI. http://www.colorado.edu/geography/gcraft/notes/intro/intro.html. Retrieved on 31 October 2011.

Roberta Pigliacampo. (2006). Multi-Tracking of Moving Objects with Unreliable Sensors for Mobile Robotic Platforms. *Master Thesis*. 1 (2), p1-12.

Ting He, Chatschik Bisdikian,Lance Kaplan,Wei Wei, Don Towsley. (2008). Multi-Target Tracking Using Proximity Sensors. *Multi-Target Tracking Using Proximity Sensors*. 1 (D), p3-6.

Stefano Coraluppi, Craig Carthel, and Mahendra Mallick. (2008). Multi-Target Tracking with Unattended Ground Sensors (UGS) Data. *Multi-Target Tracking with Unattended Ground Sensors (UGS) Data*. 1 (2), p1-11.

java2sun.com. (2011). *PDF Renderer*. Available: http://www.java2s.com/Open-Source/Java-Document/PDF/PDF-Renderer/com.sun.pdfview.htm. Last accessed 9th January 2011.

# 2. System Overview

In this section, we will give a general description of our system. First, we will describe the main goals of our system. Then we will explain interfaces that the system will provide and functionality that supported by these interfaces. Then we will look closer to the system and its sub components.

In the end of our project, we will release a software that will be used for target tracking using seismic sensors. The software will compute the data that come from sensors (virtual or real) and produce a predicted path for target. In addition, it will be able to analyze each tracking session and give out detailed report that contains graphics and diagrams.

Our system will provide three interfaces; user interface, joystick interface and sensor interface. The user interface will be the control panel of our software and it will provide some functions for managing the system and give visual feedback of these functions. The user of our software will be able to determine in which mode the software will run and to switch between these modes in run time. These modes will be of two types, “visual simulation using real sensors” and “visual simulation using virtual sensors”. In “visual simulation using real sensors” mode, our software will listen seismic sensors which will be located previously by using sensor interface and track targets relying on the data that comes from these seismic sensors. During this process, our software will also do a real-time simulation of the predicted path.

In “visual simulation using virtual sensors” mode, our software will provide an interface to user to locate virtual sensors. When the user finishes locating virtual sensors, our software will start listening joystick input using joystick interface and track target which will be controlled by user with a joystick or another joystick-like device. Like “visual simulation using real sensors” mode, our software will do a real-time simulation of the predicted path and in addition to “visual simulation using real sensors” mode, it will also simulate real path of the target hence the user will be able to see the difference between real and predicted paths.

The sensor interface will use Waspmote Gateway to communicate with seismic sensors. This interface will listen to data that come from seismic sensors continuously and forward it to our system after converting data to human-understandable form. This interface will only be used in “visual simulation using real sensors” mode.

The joystick interface will need a joystick or other joystick-like device to be connected to our system. In case of not recognition of the device, the user will be able to assign movements his/her device's keys. This interface will only be used in “visual simulation using virtual sensors” mode.

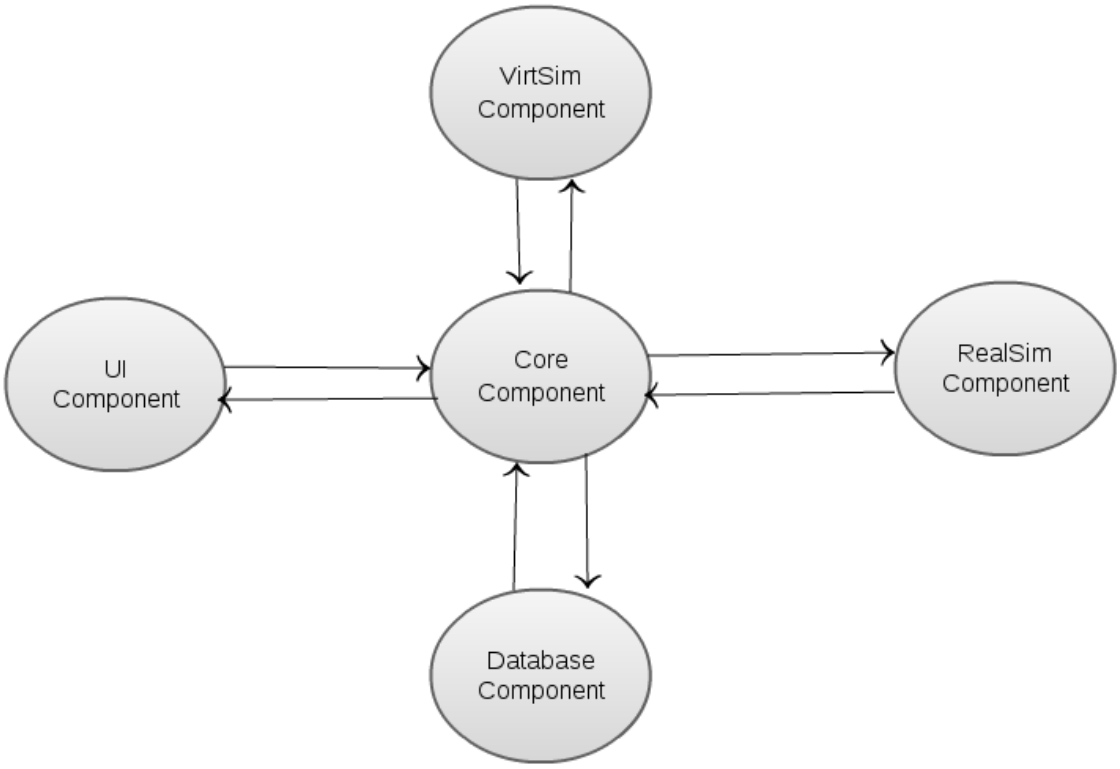



Figure 1: System Overview

Our system will consist of five main components, user interface component, virtualSim component, realSim component, database component, core component. User Interface component will manage communication between user and the system. It will provide a graphical interface to user to control the system and it will also visualize path simulations.

Database component will handle communication between system and database management system. All other components will communicate with database management system through this component.

The main purpose of the realSim component is to predict a path of target by using input data relying on the alarm levels of seismic sensors. The purpose of the virtualSim component is similar to readlSim component but it will generate the path relying on the alarm levels of virtual sensors.

The duty of core component will be to construct all of the other main components and handle communication between them. In other words the core component is operator of the system. Every component will communicate with other components through core component.

# 3. Design Considerations

In this section we will give information about our design approach. First, we will describe assumptions, dependencies and constraints of our software, then we will explain the goals of our project.

## 3.1. Design Assumptions, Dependencies and Constraints

In this section, we will provide characteristics of users of our software. Then we will explain some software and hardware dependencies of our software.

Our software is being developed as result of the request of Turkish military company ASELSAN, hence our design highly depends on their needs. This situation causes some hardware and software dependencies which we will explain later. Since the only user of our software will be ASELSAN, we do not have to worry about support a wide range of users and we can be sure of the technical knowledge of the users of our software.

In our project, we will use seismic sensors and gateways of Waspmote which are provided by ASELSAN and, because of this we will develop sensor interface of our software according to application programming interface of these Waspmote cards. The other hardware dependency of our project is joystick devices that will be used for virtual simulation. Since there is not a limitation on which device we will use, initially we will support classic analog joysticks and in case of usage of other types of joysticks, keyboards etc, we will provide a configuration interface for users where they can assign actions to their device's keys.

Our software will be developed using Java programming language, hence we will use object oriented approach for our design. The necessity of communication between the classes of our software caused a problem. We have solved this issue by deciding to use singleton pattern. The communication of our software will be established by calling related methods of target classes' instance.

The last constraint of our project occurs at developing our path generating algorithm due to insufficient number of seismic sensors. We have only one seismic sensor to try, hence we will develop our algorithm relying on the data that will come from virtual seismic sensors. The data that virtual sensors will produce will be similar to data that come from real sensor but since measurements we will do in the field may change due to air conditions, earth type etc. we do not guarantee an error-free result.

## 3.2. Design Goals and Guidelines

In this section we will explain the goals of our project and give guidelines to reach that goals.

The first main goal of our project is to generate a realistic path from the data that come from seismic sensors. As a first step, we will implement our path generating algorithm using Kalman Filter and then we will try to make our generated path error-free by changing and making additions to our algorithm. Since our path generation algorithm will work real_time, we may need to fasten our algorithm.

The second main goal is efficiency of our software. After we finish the back end programming of our software, we will optimize our code to improve efficiency. We will mainly focus on decreasing CPU usage.

The last but not primary goal of our project is to achieve simplicity for user interface. The all elements of user interface except path visualization will be implemented later and be tested for best user experience.

# 4. Data Design

## 4.1. Data Description

This section provides a description of all the data structures of the information domain of the target tracking system, including internal, global, and temporary data structures.

### 4.1.1. Internal software data structures

The data structures that are passed among the components are described below.

All the components will be able to send a Log entry (string) data to the Core component which eventually will call the method responsible of writing to the log together with the string to be written to the Log Component.

The I/O component will send the a position array from the joystick which the Core component will handle and call the function responsible for generating the path from joyStick input of the Simulator Component together with the array of positions inputed with the joystick.

The Sensor component (both real and simulated) will send alarm strength and position array to the Algorithm component.

The UI component, which is itself event-driven will send update query parameters to the Core for every user command such as adding a sensor , changing a sensor location, starting or ending a track, toggling the visibility of a layer , generating a report etc. The Core component will then call the corresponding methods of the Report and Database component.

The Report component will send the report information as two date limits and the Database will return the result set after the execution of the corresponding select query.

### 4.1.2. Global data structures

The data structures that are available to the majority of the components are described below:

The Core Class will be a singleton class, since it is appropriate to have only one instance of the Core Class and it provides a global point of access to the object.

The Log class will be a class of static functions so that all the components will be able to write the changes made by them and record them in the log.

### 4.1.3. Temporary data structures

The files created only for temporary use are the report files (pdf or doc). These files will not be persistent, unless the user selects to save the report document.

### 4.1.4. Database description

The database created as part of the target tracking application is described below:

#### *4.1.4.1. Data Entities*

The data descriptions of each of these data entities is as follows:

Alarm Data Entity

| Data Item | Type | Description | Comment |
|---|---|---|---|
| ID | Integer | ID of an alarm | |
| SensorID | Pointer | Sensor entity | Used as foreign key to the Sensor entity. |
| Strength | Float | Strength of an alarm sent | A number from 0-1024 that specifies the strength of an alarm sent. |

| | | | |
|---|---|---|---|
| TimeStamp | Date | Time of receiving the signal | |

Sensor Data Entity

| Data Item | Type | Description | Comment |
|---|---|---|---|
| ID | Integer | ID number of a sensor | |
| Location | Pointer | Position of a sensor in the map | Points to a location entity element. |
| State | Boolean | On/Off state | |

SensorLocation Data Entity

| Data Item | Type | Description | Comment |
|---|---|---|---|
| SensorID | Pointer | The sensor this position belongs to. | |
| Latitude | Float | Latitude of the selected map | |
| Longitude | Float | Longitude of the selected map | |
| XComponent | Float | X position in the map | |
| YComponent | Float | Y position in the map | |
| ZComponent | Float | Z position in the map | |

TrackLocation Data Entity

| Data Item | Type | Description | Comment |
|---|---|---|---|
| TrackID | Pointer | The track this location is part of. | |
| Latitude | Float | Latitude of the selected map | |
| Longitude | Float | Longitude of the selected map | |
| XComponent | Float | X position in the map | |
| YComponent | Float | Y position in the map | |
| ZComponent | Float | Z position in the map | |
| TimeStamp | Date | The time of the position recorded | Needed to be able to reconstruct a track after its been recorded. |

RealTrack Data Entity

| Data Item | Type | Description | Comment |
|---|---|---|---|
| ID | Integer | Id of the track input | |
| Type | Boolean | JoyStick or real input | 0 for real, 1for joystick input |
| SessionID | Pointer | Id of the session | A session is kept for a specific time interval |

GeneratedTrack Data Entity

| Data Item | Type | Description | Comment |
|---|---|---|---|
| ID | Integer | Id of the track generated | |
| SessionID | Pointer | Id of the session | A session is kept for a specific time interval |

Session Data Entity

| Data Item | Type | Description | Comment |
|---|---|---|---|
| ID | Integer | Id of the session | |
| StartTime | Date | Starting time of the session | A session is kept for a specific time interval |
| EndTime | Date | End time of the session | |

### *4.1.4.2. Entity-Relationship Diagram*

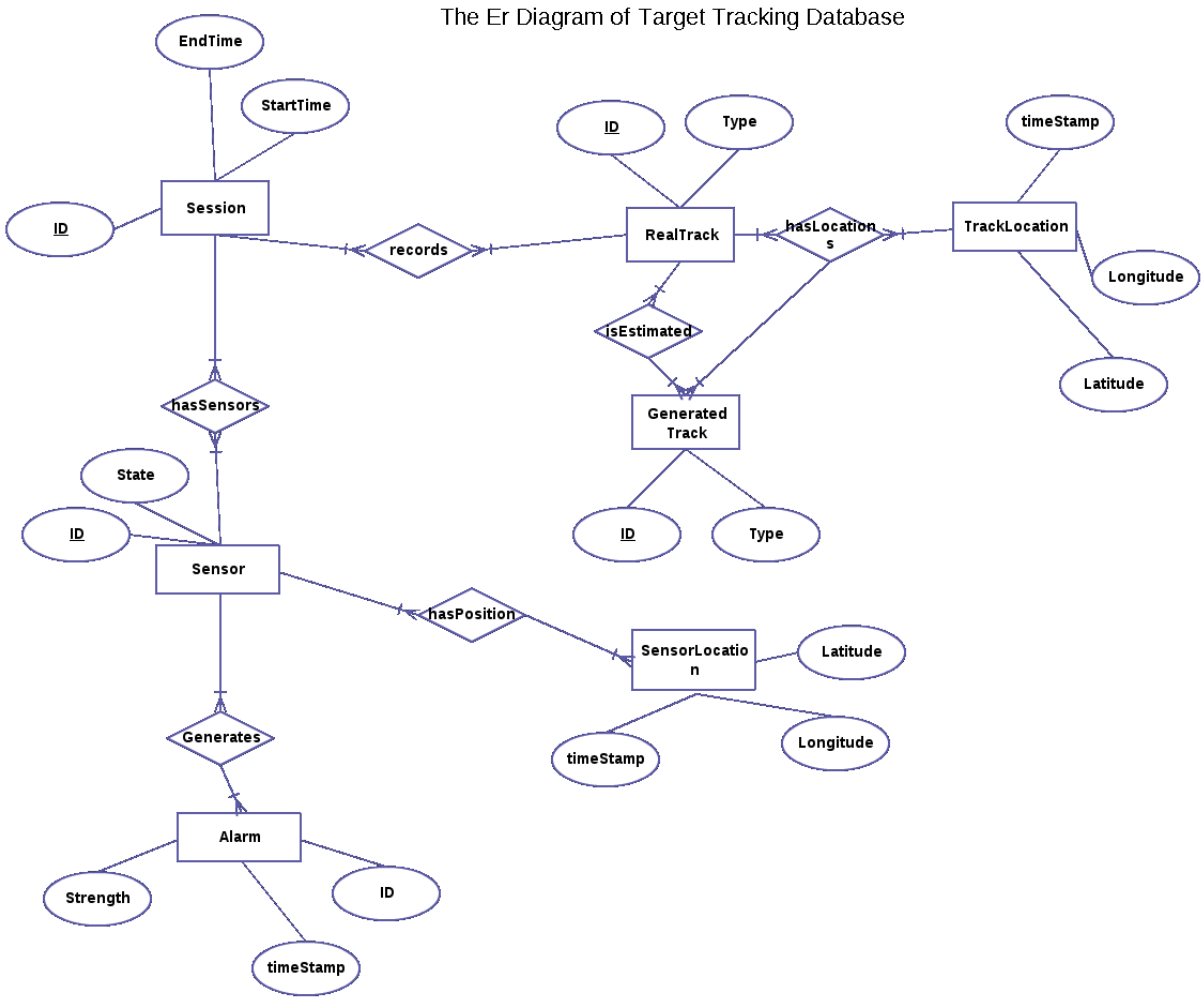



Figure 2: ER Diagram

### *4.1.4.3. Data Relationships*

The Logical Structure of the data to be stored in the Target Tracking Database is as follows:

A RealTrack or GeneratedTrack data Entity is in aggregation type of relationship with TrackLocation data Entity.

A Track(real or generated) will be composed of a continuous array of TrackLocation Objects ordered according to their timestamps.

A Sensor is in association type of relationship with Alarm entity, each Alarm is associated with a sensor id belonging to the sensor the alarm is sent from.

Finally each track, and all the information it includes is associated with a session data entity, keeping track of all the tracks recorded and generated during a specific time interval.

## 4.2. Data Dictionary

The system entities or major data along with their types and descriptions are listed in this section. If

Since we provided an OO description, we list here the objects and their attributes, the methods and the method parameters.

| Name | Type | Description | Refer to Section |
|---|---|---|---|
| layers | Component | Different layers of visibility in the user interface | 5.2.1. |
| updateSensorLayer | Method | Updates the sensor layer | 5.2.1. |
| updateTrackLayer | Method | Updates the track layer | 5.2.1. |
| displayReport | Method | Displays the generated report | 5.2.1. |
| addSensor | Method | Adds a sensor to the simulator | 5.2.1. |
| removeSensor | Method | Removes a sensor from the simulator | 5.2.1. |
| replaceSensor | Method | Replaces a sensor form the simulator | 5.2.1. |
| enableSensor | Method | Sets the sensor's state to active | 5.2.1. |
| disableSensor | Method | Sets the sensor's state to | 5.2.1. |

| | | | |
|---|---|---|---|
| | | inactive | |
| addTrack | Method | Adds a track to the simulator | 5.2.1. |
| removeTrack | Method | Removes a track from the simulator | 5.2.1. |
| startTrackRecording | Method | Starts generating a track | 5.2.1. |
| stopTrackRecording | Method | Stops generating a track | 5.2.1. |
| generateReport | Method | Generates a report for a time interval | 5.2.1. |
| ui | Component | | 5.2.2. |
| db | Component | | 5.2.2. |
| simulation | Component | | 5.2.2. |
| io | Component | | 5.2.2. |
| forwardData | Method | Forwards data between components | 5.2.2. |
| log | Method | | 5.2.3. |
| logError | Method | | 5.2.3. |
| device | Component | | 5.2.4. |
| startProcess | Method | | 5.2.4. |
| stopProcess | Method | | 5.2.4. |
| sendData | Method | | 5.2.4. |
| intervalStartTime | Data Entity | | 5.2.6. |

| | | | |
|---|---|---|---|
| intervalEndTime | Data Entity | | 5.2.6. |
| reportFormat | Data Entity | | 5.2.6. |
| getIntervalStartTime | Method | | 5.2.6. |
| setIntervalStartTime | Method | | 5.2.6. |
| getIntervalEndTime | Method | | 5.2.6. |
| setIntervalEndTime | Method | | 5.2.6. |
| getReportFormat | Method | | 5.2.6. |
| setReportFormat | Method | | 5.2.6. |
| generateReport | Method | | 5.2.6. |
| createQueryStatement | Method | | 5.2.5. |
| createUpdateStatement | Method | | 5.2.5. |
| executeQuery | Method | | 5.2.5. |
| executeUpdate | Method | | 5.2.5. |
| alarmlevel | Data Entity | Alarm level sent by the sensor | 5.2.10. ,5.2.11. |
| batterylevel | Data Entity | Battery level of the sensor | 5.2.10. |
| send_alaram_level | Method | | 5.2.10. ,5.2.9. |
| send_battery_level | Method | | 5.2.10. ,5.2.9. |
| set_alarm_level | Method | | 5.2.11. |
| get_software_sensor_alarm_level | Method | | 5.2.7. |
| send_alarm_levels_of_real_sensor | Method | | 5.2.7. |

| | | | |
|---|---|---|---|
| send_alarm_levels_of_software_sensor | Method | | 5.2.7. |
| get_location | Method | Gets the location | 5.2.7. |
| get_track | Method | Gets the generated/real track | 5.2.7. |
| current | Component | | 5.2.8. |
| previous | Component | | 5.2.8. |
| currentTrack | Component | | 5.2.8. |
| set_current_location | Method | Sets the current location | 5.2.8. |
| generate_track | Method | Path generation algorithm | 5.2.8. |

# 5. System Architecture

## 5.1. Architectural Design

We have designed our software following the singleton design pattern principles. We have developed a modular system structure.

The system's core is the Core component. The main communication among the components are handled and organized through the core. The communication is handled by the core by calling the instances of each component.

We have used aspect oriented approached in our logging system.

The pictorial presentation of the system structure is shown below.

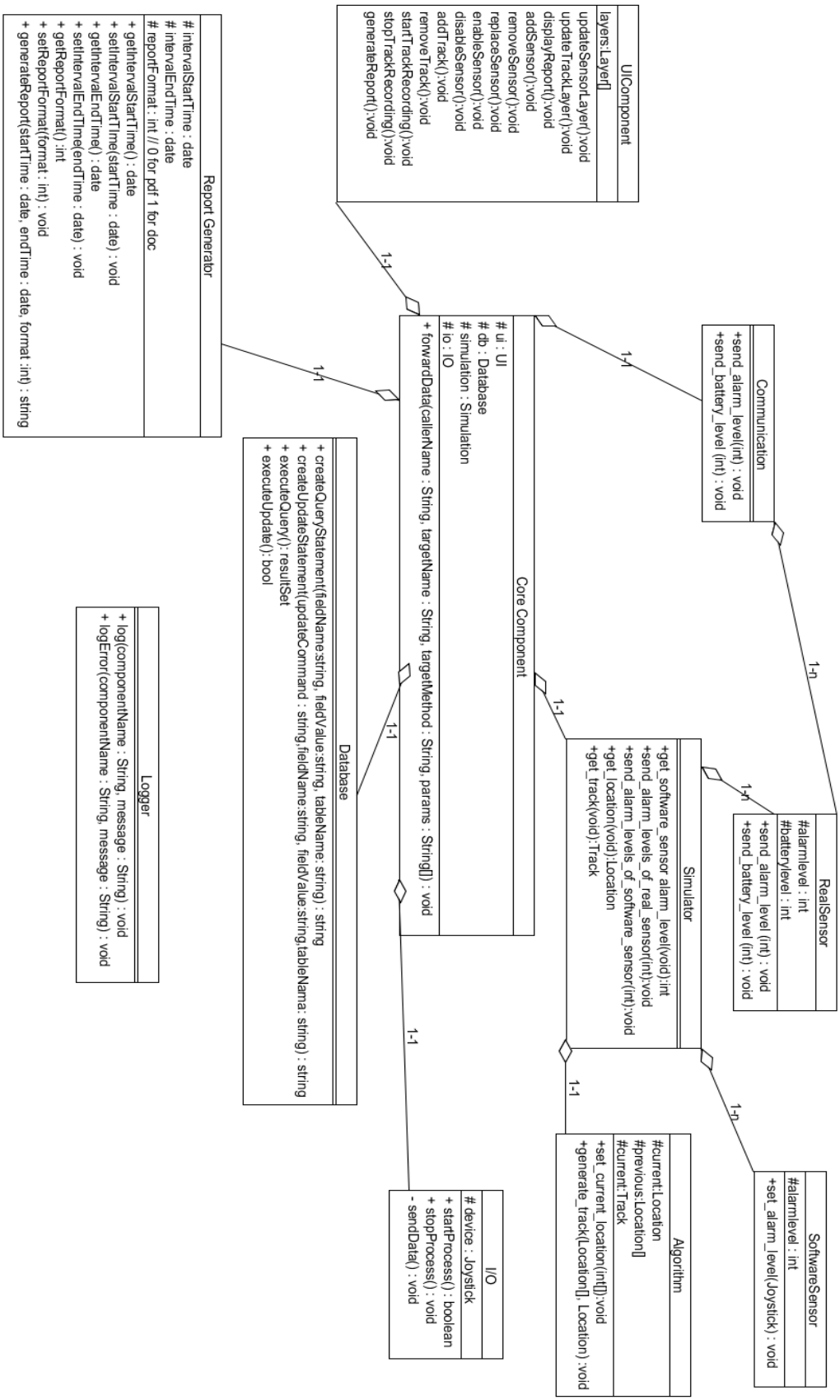



Figure 3: Component Diagram

## 5.2. Description of Components

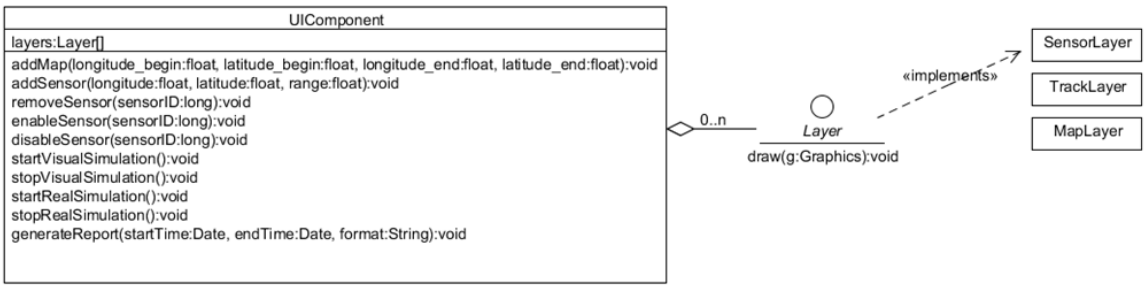



Figure 4: Class Diagram for UI Component

### 5.2.1. UI Component

The main purpose of the UI component is to handle the communication between the user and the core functions and provide a GUI.

#### *5.2.1.1. Processing Narrative for UI Component*

The functionality of the UI component is to handle the system calls from the user interface. This component includes map, sensor, track layers to show the geographic map, sensors and tracks visually. It also includes menus to call core and report functions.

This component is responsible for communicating with the Core and ReportGenerator components.

#### *5.2.1.2. UI Component Interface Description*

The UI Component has interfaces with the Core and ReportGeneartor component. The Core component uses the UI interfaces' input interfaces to update the changes in the state of the current session, such as a track, sensor or a map change. The input interfaces for the report component is for showing the generated report in the GUI.

The output interfaces of the UI Component is for calling the Core component functions, such as add a new sensor or a new track input, and for communicating with the ReportGenerator component for report generation.

**Interfaces:**

These functions will call the related functions of Core and ReportGenerator components.

addMap()

addSensor()

removeSensor()

enableSensor()

disableSensor()

startVirtualSimulation()

stopVirtualSimulation()

startRealSimulation()

stopRealSimulation()

generateReport()

#### *5.2.1.3. UI Component Processing Detail*

The layers in the UI Component contain visual representation of the current state of the program session, such as the places of the sensors, the current points in the track or the current coordinates chosen for the map. When the user triggers this component through menus, the change is reflected to the GUI and the related Core or ReportGenerator function is called.

#### *5.2.1.4. Dynamic Behavior for UI Component*

The possible use cases for the UI component:

-User clicks on add map button

-User clicks on the map to add a sensor

-User clicks on remove sensor button

-User clicks on enable/disable sensor checkbox

-User clicks on start/stop simulation button

-User clicks on generate report button.

All the use cases except the last one has the same sequence diagram between UI and Core.

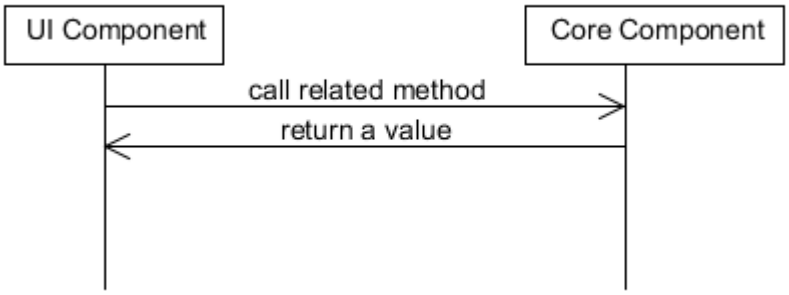



Figure 5: The sequence diagram for UI-Core interaction

The last use case has a sequence diagram from UI to ReportGenerator.

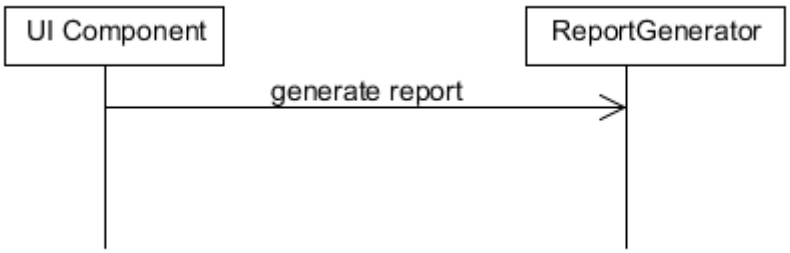



Figure 6: The sequence diagram for UI-ReportGenerator interaction

### 5.2.2. Core Component

This will be the main component of the software. It will manage communications between software components. Every component in the software except core component will communicate with each other indirectly through core component. Hence the main purpose of this component is to forward messages and requests to related components. This component will also construct other components at the beginning of the program.

#### 5.2.2.1. Processing narrative for core component

The core component will construct other components at the beginning of the program. After that, it will wait for a component which may need to communicate with another component until the end of the program. Before the program is terminated, the core component will destruct all other components and save necessary data for later use.

The communication between two components will be handled by the communication methods of the core component which are specific to each component in the system. Once a component needs to communicate with another component, it will call related method of the core component and then core component will call related method of the target component and return the data that is returned from the target component to caller component.

#### 5.2.2.2. Core Component Interface Description

The core component will be in interaction with every other component. Since it will manage the communication between system components, it will provide an interface for communication. The interface methods of the core component are listed below:

startRealSim() - notifies realSim component to start processing

stopRealSim() - notifies realSim component to stop processing

startVirtualSim() - notifies virtualSim component to start processing

stopVirtualSim() - notifies virtualSim component to stop processing

pollRealSim() - poll request for realSim component

pollVirtualSim() - poll request for virtualSim component

generateReport(startTime, endTime) – generate request for report component

#### 5.2.2.3. Core Component Processing Detail

The core component's processing details are given step by step below:

1. Construct other components

2. Wait for other component to invoke some method to communicate with other component

3. Call target component's related method

4. Return the data which is also returned from the target component's method

#### *5.2.2.4. Dynamic Behavior of Core Component*

Here are the sequence diagrams for the Core Component.

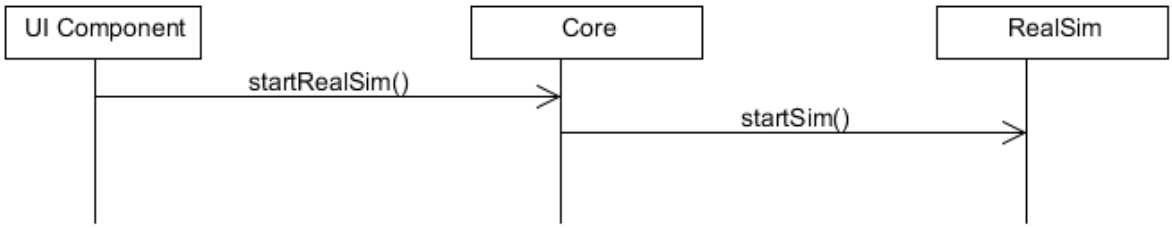



Figure 7

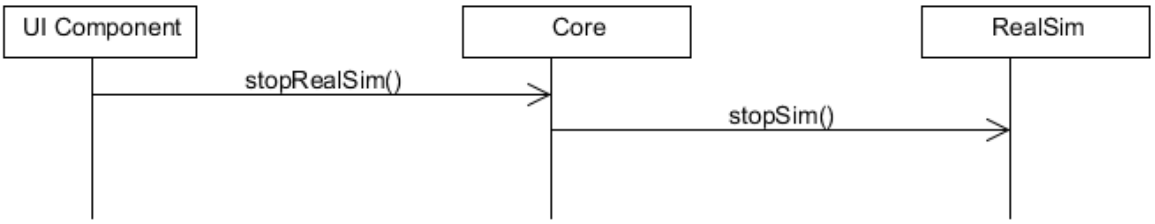



Figure 8

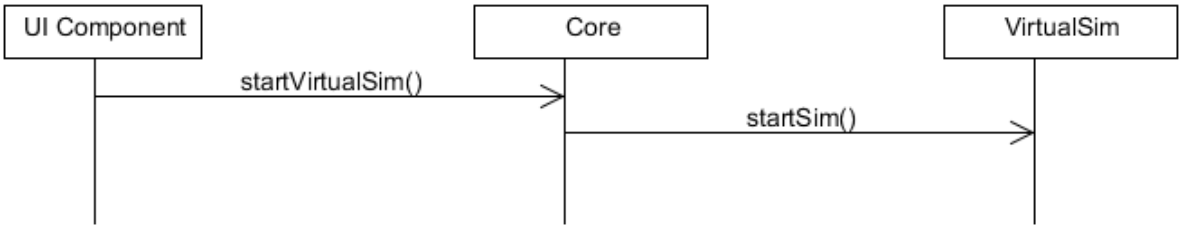



Figure 9

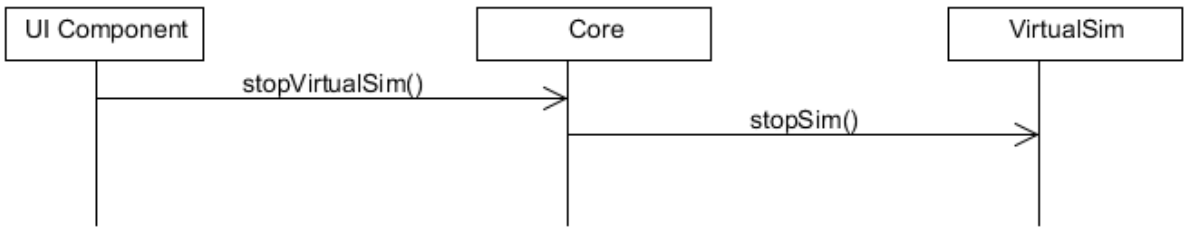



Figure 10

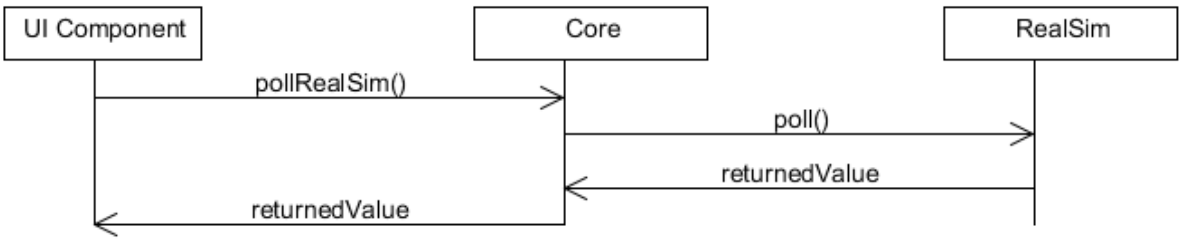



Figure 11

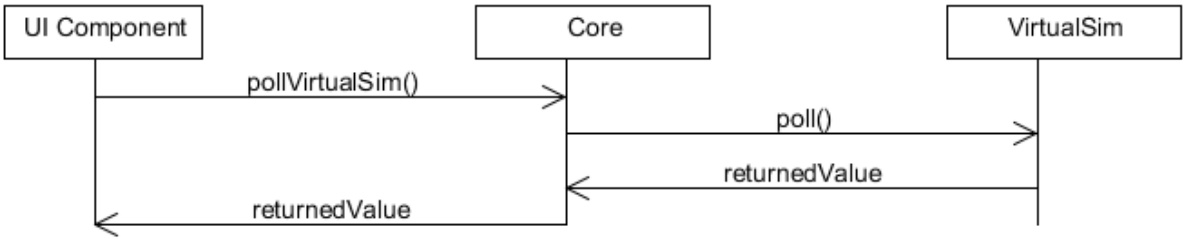



Figure 12

### 5.2.3. VirtualSim Component

The software will be able to simulate virtual path based on virtual sensors and joystick input. VirtualSim component's duty is to manage both virtual sensors and joystick input and produce a predicted path relying on the data that come from virtual sensors.

#### *5.2.3.1. Processing Narrative for VirtualSim Component*

The VirtualSim component will start simulation when it recieves a notification from core component. The component will construct input component, virtual sensor component and trackGenerator component before starting to simulation process. During simulation process, whenever core component requests genrated position (invoke poll method), the VirtualSim component will get position data from input component, use this data to update virtual sensors and use these sensors to generate path. The generated path will be used bu UI component to visualize simulation.

#### *5.2.3.2. VirtualSim Component Interface Description*

startSim() - this method will be called to start simulation process

stopSim() - this method will be called to terminate simulation process

poll() - this method will be called to get predicted path.

getInputPath() - this method will be called to get real path which is produced by joysitck.

### 5.2.3.3. VirtualSim Component Processing Detail

The virtualSim component will start waiting for poll requests when it's startSim method is invoked by core component. In simulation mode, the poll method of the VirtualSim component will be called frequently to gather predicted position. Whenever the poll method is invoked, the VirtualSim component will call poll method of input component and get changes of x axis and y axis values. Then it will compute these values and produce a new x and y position for real path to update virtual sensors' alarm level. After that the trackGenerator component will be used to generate an estimated path relying on the alarm levels of sensors. Finally, the estimated path will be returned to the invoker.

### 5.2.3.4. Dynamic Behavior of VirtualSim Component

Here are the sequence diagrams for the VirtualSim component.

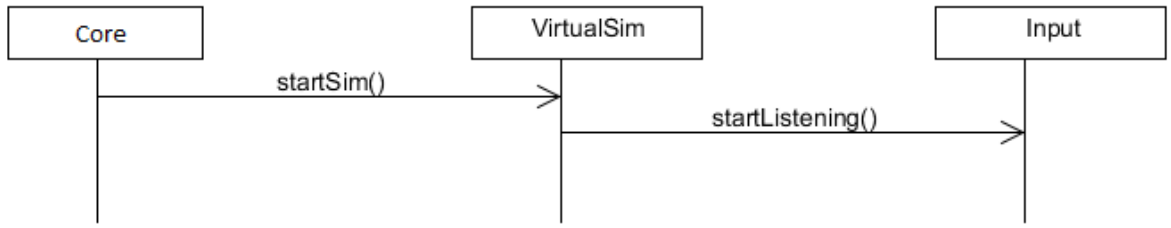



Figure 13

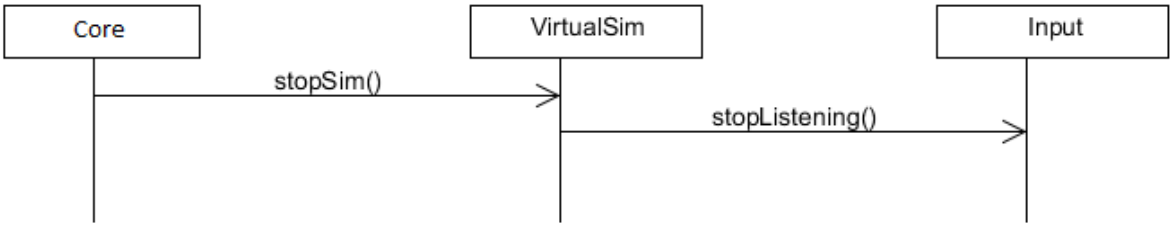



Figure 14

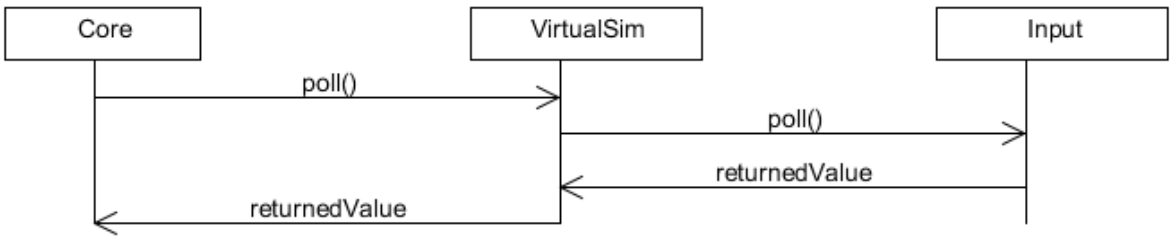



Figure 15

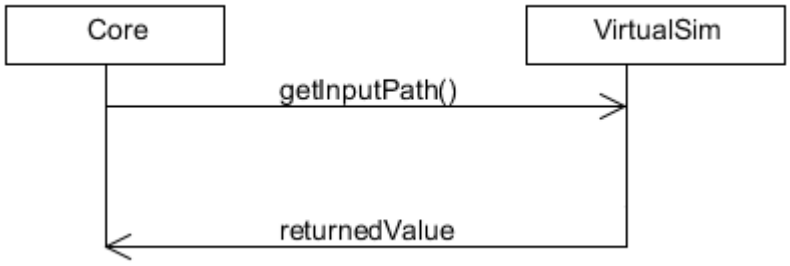



Figure 16

### 5.2.4. Input Component

The software will be able to listen input from a joystick-like device to simulate real path when working with virtual sensors. The input component is a layer between joystick device and VirtualSim component. When the software is running in virtual mode, this component will wait for input from joystick-like device and convert it into human understandable form for VirtualSim component.

#### *5.2.4.1. Processing narrative for input component*

The input component will be constructed by virtualSim component. When requested, the input component will scan computer for available devices and return a list of them to user to choose one. The input component will start listening to selected device when it receives a notification from VirtualSim component. For each time that poll method invoked, the input component will return the changes in x and y axis to VirtaulSim component.

#### *5.2.4.2. Input Component Interface Description*

The input component will be in interaction with VirtualSim component and UI component. The interface methods of the input component are given below:

getDeviceList() - this method will return the list of available devices

setActiveDevice(name) – this method will mark the specified device for listening

poll() - this method will return the changes in x and y axis

startListening() - this method will start listening from selected device.

stopListening() - this method will stop listening from selected device.

### 5.2.4.3. Input Component Processing Detail

The input component will use Jinput library to interact with joystick device. The component will get button information – which button the user pressed – and will compute the changes in x and y position. When poll method of the input component is invoked, it will return the the total change in x and y positions and reset the change values.

### 5.2.4.4. Dynamic Behavior of Input Component

Here are the sequence diagrams for Input component.

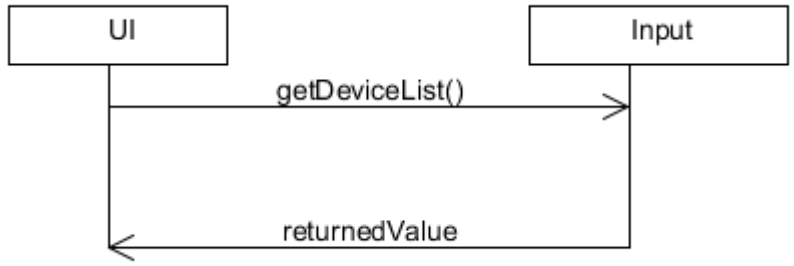



Figure 17

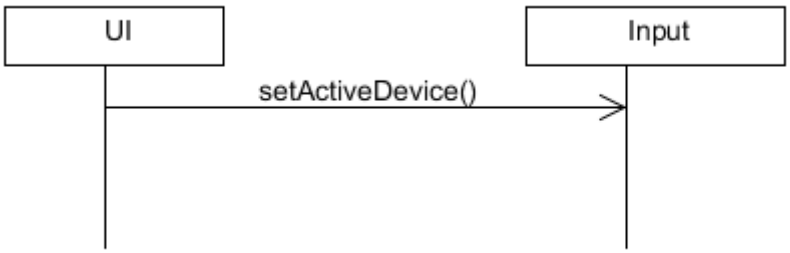



Figure 18

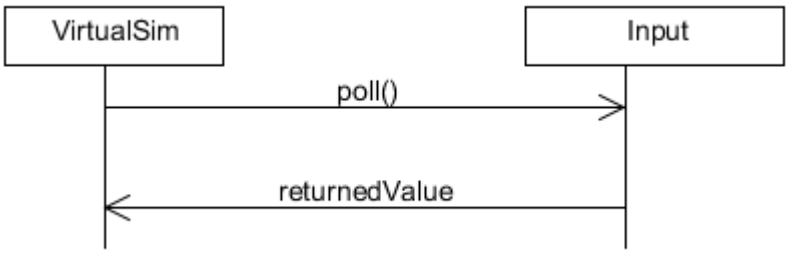



Figure 19

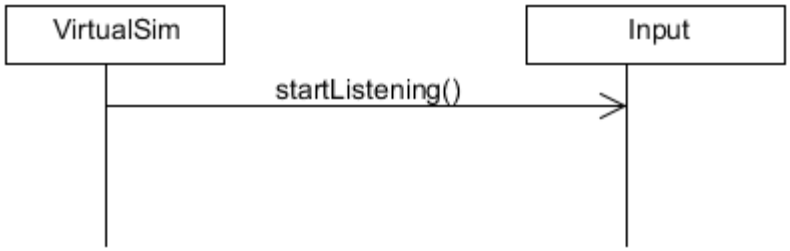



Figure 20

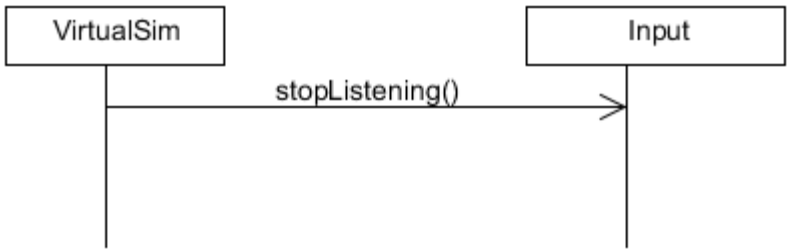



Figure 21

### 5.2.5. TrackGenerator Component

The TrackGenerator component is the most important component of the system. It will predict the target's path by using the sensor's alarm level and apply Kalman filter on this result to make prediction to be more realistic. The TrackGenerator component will be both used by VirtualSim component and RealSim component.

#### *5.2.5.1. Processing Narrative for TrackGenerator Component*

The TrackGenerator will produce path in two stages. At the first stage the trackGenerator component will estimate a point of the target relying on the alarm level and position of the sensors. At the second stage the trackGenerator component will apply kalman filter on this point and previously generated points and return the estimated path.

#### *5.2.5.2. TrackGenerator Component Interface Description*

The trackGenerator component will be used by realSim and virtualSim component. The only interface method of this component is provided below:

generate(sensors) – this method will generate an estimated point

#### *5.2.5.3. TrackGenerator Component Processing Detail*

The trackGenerator component will start processing when realSim component or virtualSim component invoke it's generate method. The track generation will be completed in two steps. In the first step, the component will use alarm level and position of the sensors to predict the new location

of the target. The new location of the target will be computed by taking weighted average of the sensors' position. In the second step, the trackGenerator component will apply kalman filter to produced points including the recent one and generate the ultimate path.

#### *5.2.5.4. Dynamic Behavior of TrackGenerator Component*

Here are the sequence diagrams for TrackGenerator component.

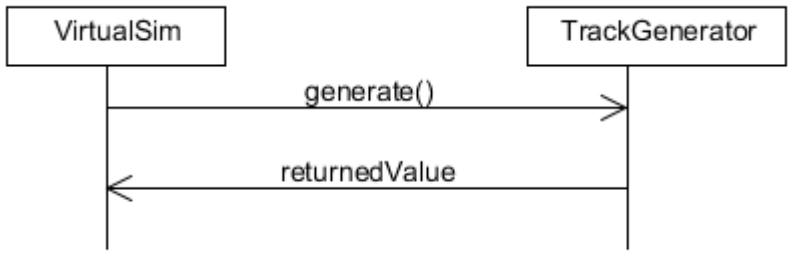



Figure 22

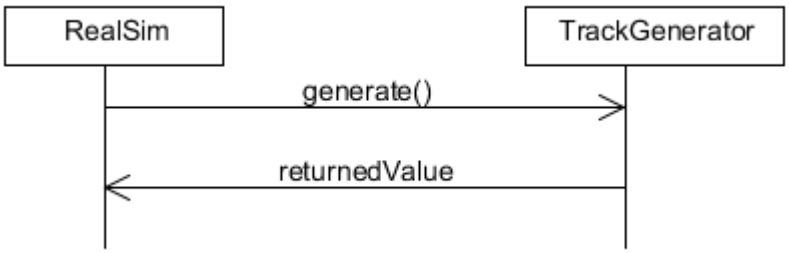



Figure 23

### 5.2.6. Logger Component

The logger component of the system will be used to log every action and error while the program is running. It will be a standalone component and every component of the system will use it.

#### *5.2.6.1. Processing Narrative for Logger Component*

The logger component will log two type of data: Actions and errors. When a component needs to log an action, the logger component will open log file of the current session and log the component's action with adding current date time. When an error occurs in system, the log component will open error log file and log the details of error with adding current date, current time and current session.

#### *5.2.6.2. Logger Component Interface Description*

The logger component will be in interaction with every component of the system. The interface methods of the logger component are provided below:

log(sessid, callerComponent, message) – will open log file of the current session and log the message

logError(sessid, callerComponent, errorMessage) - will open erro log file and log the errorMessage with current session id

#### *5.2.6.3. Logger Component Processing Detail*

When the logger component receive a log request, it will open the related file (session file for action logging and error file for error logging) and log the message with current date and time. If the request is for error logging, the logger component will also add information about current session id.

#### *5.2.6.4. Dynamic Behavior of Logger Component*

Here are the sequence diagrams for Logger component.

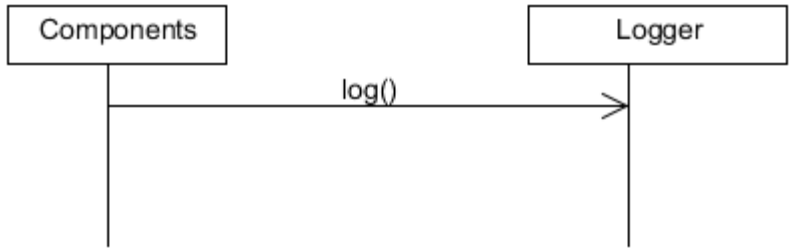



Figure 24

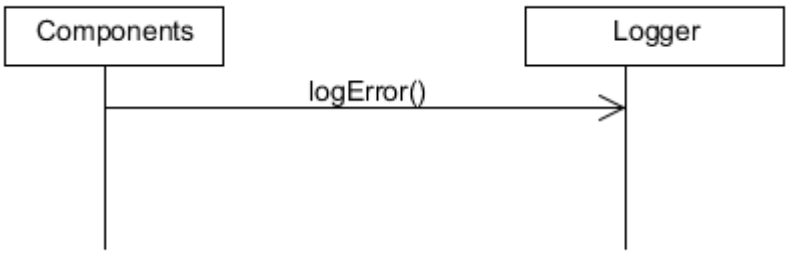



Figure 25

### 5.2.7. Sensor Component

This component is designed to get information about target's position. We have two types of sensors in our system. First type is 'real sensor' which is a hardware device. Second one is 'virtual sensor' which is a class implemented by us. This component provides our system these sensor classes.

#### *5.2.7.1. Processing narrative for Sensor Component*

Sensor Component communicates with the system and gives information about target's position in constant time interval. This time interval is  10 ms for our system. Sensor Component is responsible from setting its alarm level in this time.  This alarm level shows the distance between the sensor and target's location at a specific time. In other words, alarm level shows where the target is according to the sensor.

### 5.2.7.2. Interface description for Sensor Component

Sensor Component has interface with RealSim and VirtualSim Components. The functionality of this interface provides ability to inform the system for Sensor Component.

### 5.2.7.3 Processing detail for Sensor Component

In 'real simulation mode', system will use real seismic sensors. A real seismic sensor communicates with the system using Communication Component. When there is a vibration, it detects and generates an alarm signal. RealSim Component gets this signal in every 10 ms. In 'virtual simulation mode', system will use virtual sensors. In this mode, user gives input using I/O device. A virtual sensor is responsible from setting its alarm level according to the position of the given input. In this mode, VirtualSim Component gets this signal in every 10 ms.

### 5.2.7.4. Dynamic Behavior

Trigger: Motion of target,

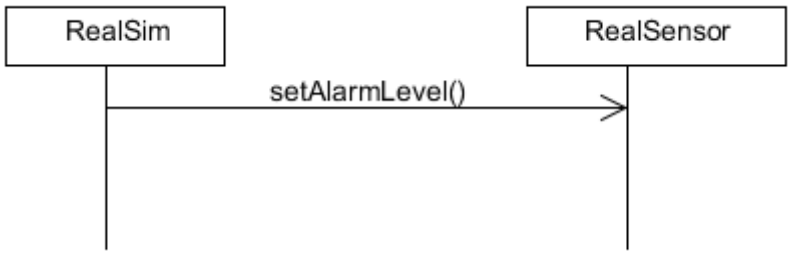



Figure 26

Trigger: I/O given by user,

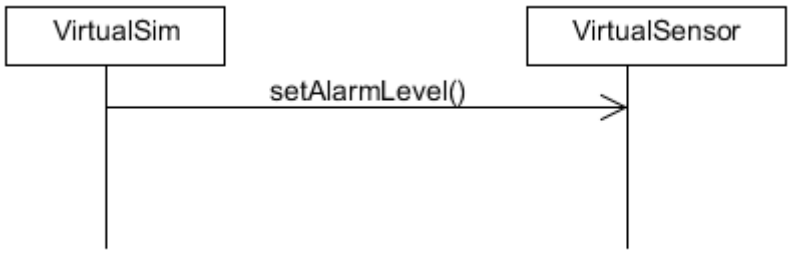



Figure 27

Two types of sensors in the system:

| **RealSensor** |
| --- |

| |
|---|
| # sensorId : int<br># location : Point<br># range : float<br># alarmlevel : int |
| + getSensorId () : int<br>+ getLocation () : Point<br>+ getRange () : flaot<br>+ getAlarmLevel () : int<br>+ setSensorId(value : int) : void<br>+ setLocation(value : Point) : void<br>+ setRange(value : float) : void<br>+ setAlarmLevel(value : int) : void |

| **VirtualSensor** |
|---|
| # sensorId : int<br># location : Point<br># range : float<br># alarmlevel : int |
| + getSensorId () : int<br>+ getLocation () : Point<br>+ getRange () : flaot<br>+ getAlarmLevel () : int |

+ setSensorId(value : int) : void

+ setLocation(value : Point) : void

+ setRange(value : float) : void

+ setAlarmLevel(value : int) : void

+ calculateAlarmLevel(value : Point) : Point

### 5.2.8. RealSim Component

RealSim Component is designed to simulate the target tracking algorithm by using real seismic sensors. In 'real simulation mode', system will this Component. In this mode, system will simulate tracking of a moving target within an area where seismic sensors are previously deployed. System will calculate the route of the target based on the detection from the seismic sensors.

#### *5.2.8.1. Processing Narrative for RealSim Component*

RealSim Component has a reference of TrackGenerator. It also gets the states of real sensors from Communication Component. So, when user wants to start the real simulation mode, it starts to listen all sensor's state. Moreover, It gets the path of the target using TrackGenerator Component.

#### *5.2.8.2. Interface Description for RealSim Component*

The interface of the Component provides following properties:

RealSim Component has interface with Communication Component so that it can listen to real sensors.

RealSim Component has interface with TrackGenerator Component so that it can get the path of the target calculated.

RealSim Component can start simulation. It means that Communication Component will start listening to sensors, TrackGenarator will start to calculating the route of the target.

RealSim Component can stop simulation. After that, Communication will stop listening and TrackGenarator will stop calculating.

RealSim Component has interface with Core Component so that it can send the generated track to UI or Database Component of the system.

#### 5.2.8.3. Processing Detail for RealSim Component

When system is in 'real simulation mode', a real seismic sensor detects and generates an alarm signal when there is a vibration. Communication Component gets the states of real sensors and sends to RealSim Component in every 10ms. After getting the states of the sensors, it sends them to the TrackGenerator to calculate the path of the target.

#### 5.2.8.4. Dynamic behavior RealSim Component

Trigger: user clicks on start 'real simulation mode'

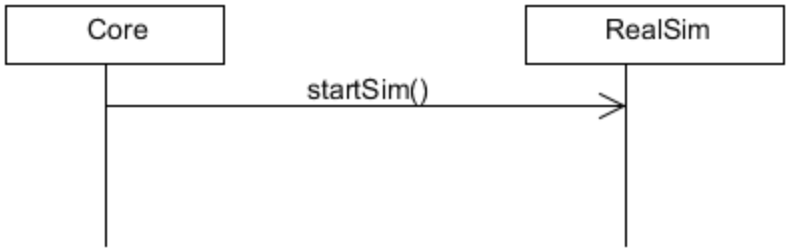



Figure 28

Trigger: user clicks on stop 'real simulation mode'

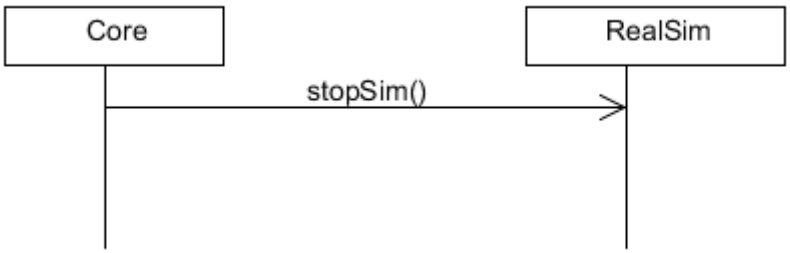



Figure 29

Trigger: system clock ticks in every 10ms

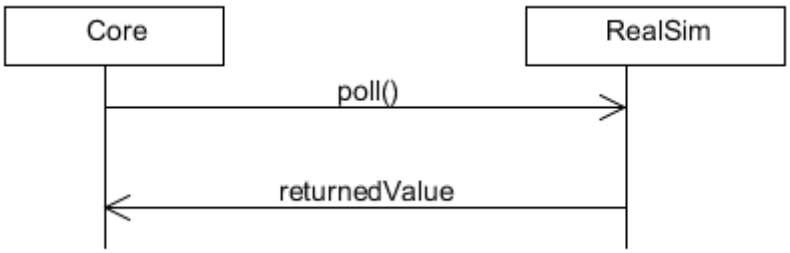



Figure 30

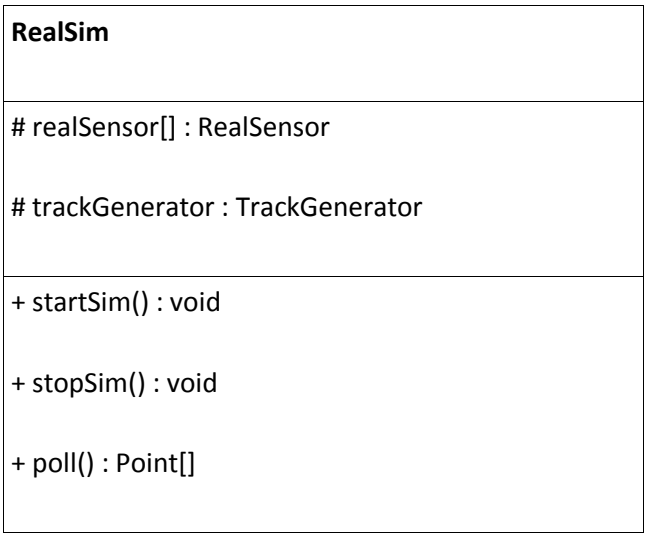



Figure 3: Class Diagram

### 5.2.9. Communication Component

It is a hardware device that communicates with real seismic sensors and RealSim Component. RealSim Component calls this device to get data from real sensors.

#### *5.2.9.1. Processing narrative for Communication Component*

When there is a target on field, a real sensor detects motion of this target. The real sensor generates an alarm signal. Then, Communication Component gets this signal in a spesific time interval and sends it to RealSim Component.

#### *5.2.9.2. Communication Component interface description*

Comunication Component has interface with RealSim and Sensor Components. This interface provides the functionality so that communication Component gets the states of the sensors. Moreover, it can send the states of the real sensors and their location information to RealSim Component.

#### *5.2.9.3. Communication Component processing detail*

In 'real simulation mode', there are many seismic sensors deployed on field. They detect any vibration on that area and generate alarm signal showing the strength of the vibration. At this point, system needs a component to get these alarm levels of the sensors in a constant time intervals. Communication Component of our system, meets that need. It gets the alarm level of every sensors and send to the RealSim Component.

#### *5.2.9.4. Dynamic behavior Communication Component*

Trigger: system clock ticks in every 10ms

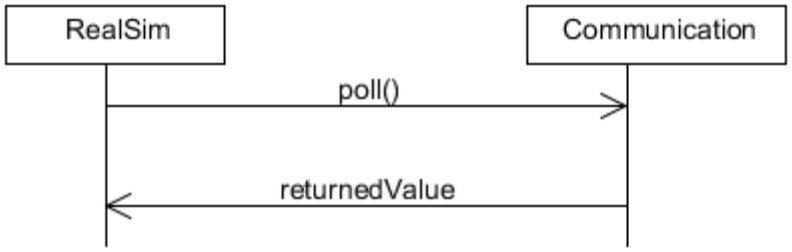



Figure 31

| **Communication** |
| --- |
| # realSensor[] : RealSensor |
| + getRealSensor() : realSensor[]<br><br>+ poll() : void |

Figure 4: Class Diagram

### 5.2.10. Database component

The Database component is described in the section below, together with its responsibilities, input and output interfaces, processing details, dynamic behavior and the corresponding diagrams (class, interaction etc.)

#### *5.2.10.1. Processing narrative for Database component*

The purpose of the Database component is to organize the database in order to provide an efficient way of accessing the data items of the application. This component primarily serves to the Report Generator component and the Core Component.

The Report Generator component will use the Database component in order to retrieve the information required to generate a report.

The Core component will send the Database component the entities to be stored in the database, such as sensors, alarms, real and generated tracks, etc.

#### *5.2.10.2. Database component interface description*

The Database component will not have any modeling attributes. Instead, it will have the intuitive createQueryStatement(), createUpdateStatement(), executeQuery(), executeUpdate(). Besides, there will be some specific and more frequent queries such as storeSessionInfoUpdate(), storeSensorInfoUpdate(), storeRealPathUpdate(), storeGeneratedPathUpdate().The reason of having these functions that will perform specific queries is because of the frequency these queries will be

made. The input to these methods will be provided by the Core and Report Generator component. The core component will input the entities created during the application's running time. Each sensor along with its location, and the alarms it sends, and the tracks generated using this information are stored in the database.

The Database component will output a result set at the end of the execution of any update/query statement.

#### *5.2.10.3. Database component processing detail*

The Database component has no subordinates. Hence a single class diagram for the Report Generator component is provided below:

| **Database** |
| --- |
| |
| +storeSessionInfoUpdate( sessionID : int,startTime : date, endTime : date) : int<br>+storeSensorInfoUpdate( sessionID : int, Sensor[] sensors) : int<br>+storeRealPathUpdate( sessionID : int, Point[]) : int<br>+storeGeneratedPathUpdate (sessionID : int, Point[]) :int<br>+createQueryStatement(fieldName : string, fieldValue : string, tableName : string) : string<br>+createUpdateQuery(updateCommand : string, fieldName : string, fieldValue : string, tableName : string ) : string<br>+executeQuery():resultSet<br>+executeUpdate():int |

There are some restrictions/limitations for Database component. PostgreSQL will be used as the object-relational database system since it allows storage of position items as longitude and latitude and it also strongly conforms to the ANSI-SQL:2008 standard.

Very frequent join operations performed during the queries required for the report generation is a performance issue for the Database component. In order to improve the efficiency of this operation, a sessionID foreign key will be present in all of the tables that store information about the tracking in a specific time interval. Another performance issue, the one regarding the frequency of database

storage queries (storeSessionInfoUpdate(), storeSensorInfoUpdate(), storeRealPathUpdate(), storeGeneratedPathUpdate()) is improved by performing these queries in specific functions.

#### 5.2.10.3.5. Processing detail for each operation of Database component

The processing details (narrative and algorithmic) for the operations realized by the Database component are as follows:

The user creates/adds/removes a sensor, changes the location of a sensor from the UI component. The changes information is sent to the Core Component.

The Core component calls the storeSensorInfo(sessionID, Sensor[]) of the Database component.

The Database component calls the executeUpdate() method and executes the created update statement that updates the Sensor information. An integer showing the number of the rows and the name of the table affected is returned to the Core Component.

Or alternatively,

The user starts/ends the track generation from the UI component. The start and end time of the track generation are sent to the Core Component.

The Core component calls the storeSessionInfoUpdate(sessionID, startTime, endTime) of the Database component.

The Database component calls the executeUpdate() method and executes the created update statement that inserts information about the new session. An integer showing the number of the rows and the name of the table affected is returned to the Core Component.

Or alternatively,

The user starts/ends the track generation from the UI component in the VirtualSim mode. The track generated by the input device and the one generated by the Track generator component are stored and sent to the Core Component as a Point[].

The Core component calls the storeRealPathUpdate(sessionID, Poin[]) and storeGeneratedPath(sessionID, Point[]) of the Database component.

The Database component calls the executeUpdate() method and executes the created update statement that inserts information about the new tracks. An integer showing the number of the rows and the name of the table affected is returned to the Core Component.

Or alternatively,

The user decides to generate a report for a specific time intervalfrom the UI component. The report information is sent to the Report Generator Component.

The Report Generator component calls the createQueryStatement and as a result an select query is created according to the sessionIDs of the time intervals for which a report is to be generated.

The Database component calls the executeQuery() method and executes the created select statement. A result set containing the rows and the name of the table created is returned to the UI Component.

#### *5.2.10.4. Dynamic Behavior for Database component*

A description of the interaction of the classes is presented. A sequence diagram is presented for the Generate Report use case realized by this component.

##### 5.2.10.4.1. Interaction Diagrams

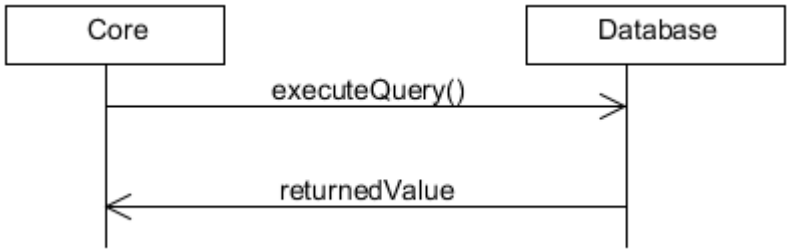



Figure 32

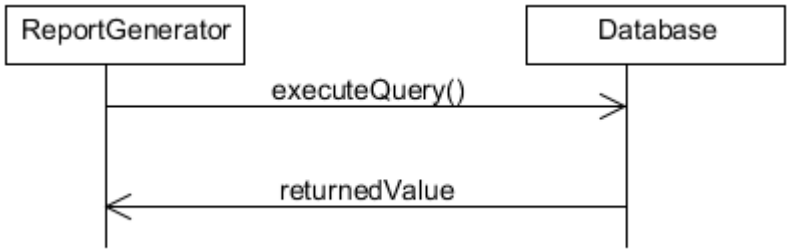



Figure 33

### 5.2.11. Report Generator component

| **Report Generator** |
| --- |

| |
|---|
| #intervalStartTime : date<br><br>#intervalEndTime : date<br><br>#reportFormat : int //0 for pdf 1 for doc<br><br>#generatedReportXML : file //xml format |
| +getIntervalStartTime() : date<br><br>+setIntervalStartTime(startTime : date) : void<br><br>+getIntervalEndTime() : date<br><br>+setIntervalEndTime(endTime : date) : void<br><br>+getReportFormat() : int<br><br>+setReportFormat(format : int) : void<br><br>+generateReport(startTime : date, endTime : date, format : int) : file<br><br>+saveGeneratedReport(filePath : string, fileName : string) |

Above is the class diagram for the Report Generator component. The Report Generator component is described in the section below, together with its responsibilities, input and output interfaces, processing details, dynamic behavior and the corresponding diagrams (class, interaction etc.)

#### *5.2.11.1. Processing narrative for Report Generator component*

The Report Generator component is responsible for the report generating functionality provided to the user.

As described in the SRS, this feature enables the user to view all relevant information of a session, after he/she provides the information related to the report to be generated in the form prompted to him and a document containing information retrieved by the database such as the map together with the sensor locations, the log of the alarms, and the real and estimated tracks so that one can evaluate the correctness of the path generation algorithm.

#### *5.2.11.2. Report Generator component interface description*

The Report Generator component will get as input from the user the start and end time of the interval for which the report is to be generated, and the format of the report. The output will be a PDF of DOC file displaying the information about the selected time interval, plotting all the generated tracks along with the real tracks corresponding to that time interval.

For all practical purposes, this component is modeled by the three attributes provided by the user as input, i.e. intervalStartTime, intervalEndTime, and reportFormat. These attributes will be protected and they will have getter and setter methods.

#### *5.2.11.3. Report Generator component processing detail*

An algorithmic description of the Report Generator component is provided below.

The Report Generator component has no subordinates. Hence a single class diagram for the Report Generator component is provided below:

| **Report Generator** |
| --- |
| #intervalStartTime : date<br><br>#intervalEndTime : date<br><br>#reportFormat : int //0 for pdf 1 for doc<br><br>#generatedReportXML : file //xml format |
| +getIntervalStartTime() : date<br><br>+setIntervalStartTime(startTime : date) : void<br><br>+getIntervalEndTime() : date<br><br>+setIntervalEndTime(endTime : date) : void<br><br>+getReportFormat() : int<br><br>+setReportFormat(format : int) : void<br><br>+generateReport(startTime : date, endTime : date, format : int) : file |

+saveGeneratedReport(filePath : string, fileName : string)

There are restrictions/limitations for Report Generator component. The format of the report can either be PDF or DOC. The time interval selected should not exceed one day time interval and should not be shorter than a session time (the time for a track to be started, ended, and estimated).

#### 5.2.11.3.5. Processing detail for each operation of Report Generator component

The processing details (narrative and algorithmic) for the report generation operation realized by the Report Generator component are as follows:

1.The user inputs the start and end time of the interval of the report, as well as the format of the document from the UI component.

2.The Report Generator component receives these parameters and calls the generateReport(startTime, endTime, format) method.

3.The generateReport() method performs a query to the database in the Database component and selects all the information available in all the tables by using the sessionIDs, their starting and ending times to join them. A result set is returned from the Database to the Report Generator component.

4.A PDF/DOC document is generated using the information retrieved from the database present in the result set returned form the generateReport() method.

5. The user is asked if he/she wants to save the generated report document.

6. If the user decides to save the document he is asked to provide a name and a path.

### *5.2.11.4. Dynamic Behavior for Report Generator component*

A description of the interaction of the classes is presented. A sequence diagram is presented for the Generate Report use case realized by this component.

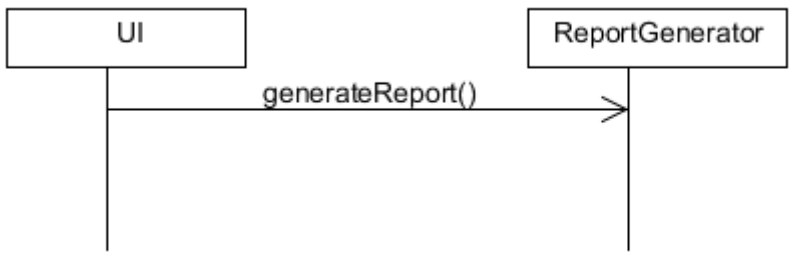



Figure 34

# 6. User Interface Design

In this section, the design of the user interface will be explained.

## 6.1. Overview of User Interface

The user interface is composed of two main screens and helper screens.

The first screen is the entrance screen (Figure 35). The user is directed to the system functions from this screen. This screen has three options for the user: Track generation with real sensors, track generation with virtual sensors, report generation, respectively. When the user clicks a button, the user is redirected to another screen.

When the user clicks on the first choice, an options screen comes up for the user to enter latitude and longitude settings for the map (Figure 36). When the user clicks on 'Start' from this screen, the main screen of the program is opened, which consists of sensor list and map, on which the sensors and tracks are drawn (Figure 38).

Similarly, when the user clicks on the second choice, an options screen comes up for the user to enter latitude and longitude settings for the map (Figure 37). This screen also has a list to choose the input method. When the user clicks on 'Start' from this screen, the main screen, similar to the one described above, opens (Figure 38).

The third choice brings the report generation option screen, which contains 'from date', 'to date', 'report type' options, and a 'Generate' button (Figure 39).

## 6.2. Screen Images

Here are some screenshots from the program.

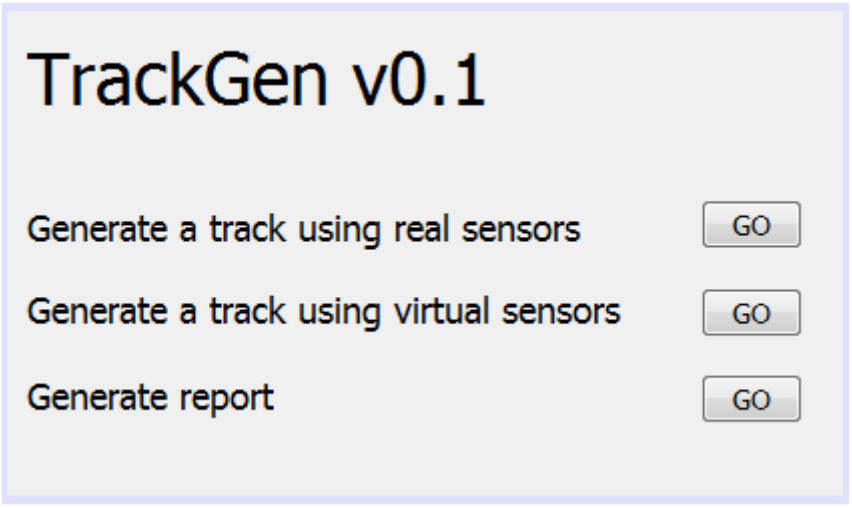



Figure 35: Entrance Screen

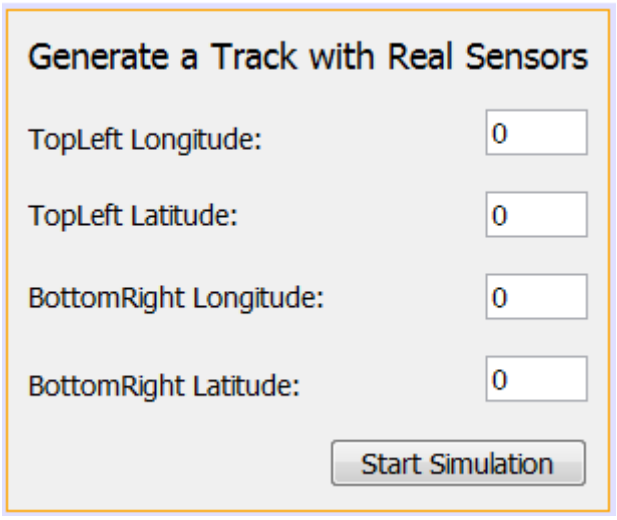



Figure 36: Generate a Track with Real Sensors Menu

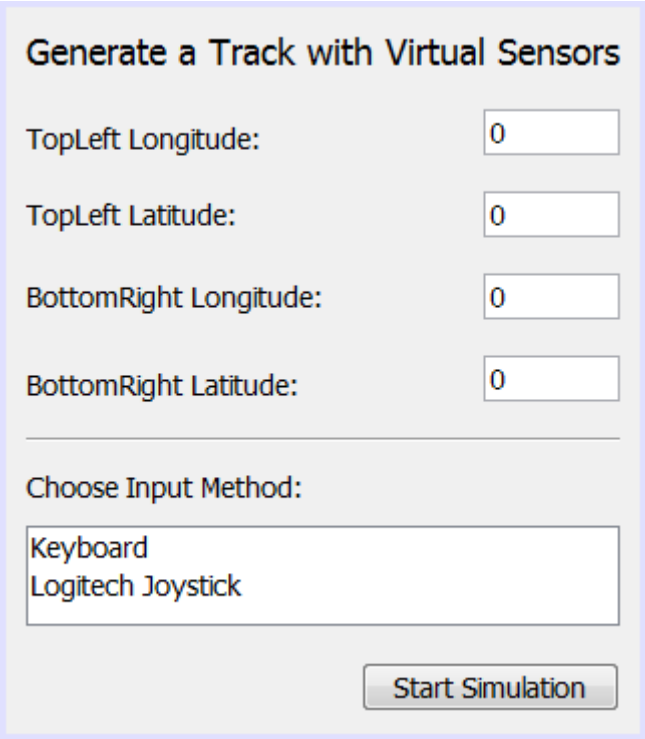



Figure 37: Generate a Track with Virtual Sensors Menu


Figure 38: Main Screen

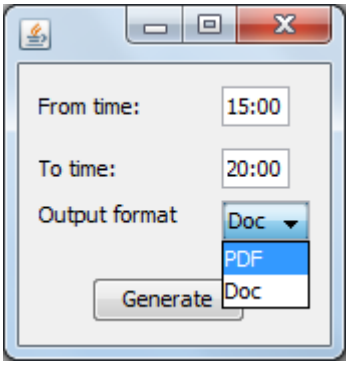



Figure 39: Generate Report Screen

## 6.3. Screen Objects and Actions

In this section, the screen objects and their actions will be explained.

Entrance screen (Figure 4): There is one label for software name. 3 labels for options and three buttons with label “GO”. When the user clicks a button, the related screen opens, as described above.

Generate a Track Menu (Figure 5): There are labels for explanations, text fields for input data, and button for starting the simulation. When the button is clicked, main screen (Figure 7) is opened.

Generate a Track Menu (Figure 6): There are labels for explanations, text fields for input data. There is a list for list of input devices. There is a button for starting the simulation. When the button is clicked, main screen (Figure 7) is opened.

Main Screen (Figure 7): There is a list for sensor list, and a button for “Remove Sensor”. When this button is clicked, the related function for deleting the sensor is called.

There is a panel for “Sensor Information”. In this panel, there is a checkbox for “Enabled”. When the checkbox is enabled, the sensor is enabled. When the checkbox is disabled, the sensor is disabled from the simulation.

There is another panel for containing “Map”. There are three toggle buttons for this panel: Toggle Sensor Layer, Toggle Track Layer, Toggle Map Layer. When they are toggled on, the related layer is visible. Similarly, when they are toggled off, the related layer is invisible.

There is a button for stopping the simulation. When this button is clicked, the program goes back to the main screen.

Generate Report (Figure 8): There are two text boxes for entering time values. There is a combo box for choosing the file type. There is a button for generating the report. When the user clicks on the button, the report is generated. Then, the program returns to the main screen.

## 7.1. UI Component

### 7.1.1. Classification

UI Component is a module of the system.

### 7.1.2. Definition

The purpose of this component is to provide interface functions between the user interface and the system.

### 7.1.3. Responsibilities

The UI component is responsible for interfacing the system functions with the user interface.

One responsibility of the UI component is to call system functions according to the user interface interaction. When the user interacts with the system through the user interface, related calls are made to the UI component. The UI component calls functions of the related components, according to the user request.

Another responsibility of the UI component is to reflect changes made by the system back to the user interface.

The system works with polling system. The UI component should call poll function of the core component in a fixed time period regularly to update the user interface and system.

### 7.1.4. Uses/Interactions

This component serves as a bridge between the user interface and the Core, ReportGenerator components.

When a user clicks on a button for adding a map or a sensor, this component calls related functions from the Core component and reflects changes back to the UI. Similarly, when the user clicks for report generation, the functions of the ReportGenerator component called by this component.

The other components don't call functions of this component. This component calls functions of the Core and ReportGenerator component in fixed time interval or on demand by the user. Then, this component updates the user interface, according to the return values of the function calls.

### 7.1.5. Processing

This component mainly works event-based. When the user interacts with the interface, a function of this component is called by the event-handler.

poll(): This function is called once in every 10 milliseconds. This function is used for polling the system, including the input devices and the software/hardware sensors. The poll() function calls the poll() function of the core component and initializes the polling chain.

### 7.1.6. Interface/Exports

This component has an interface to the user interface.

addSensor() and removeSensor(): When the user clicks on the map, a sensor is added on the pixel the user clicked. Similarly, removeSensor() is called when the user clicks on the removeSensor() button. Mouse clicks generate on-click events of swing components and the related event-handlers are called, which in turn call addSensor or removeSensor.

enableSensor() and disableSensor(): When the user clicks on the checkbox of the sensor to enable/disable it, these methods will be called. They will call the related method of the core component will be called to fulfill the task.

When the user clicks on add map button, the addMap() method is called. This method uses the worldwind library[1] to show the geographic photo of the area, whose latitude and longitude borders are given by the user.

Similarly, when the user clicks on generate report button, generateReport() method is called, with document type and start/end date parameters as arguments.

StartVirtualSim/stopVirtualSim/startRealSim/stopVirtualSim: These methods are called when the user starts/stops the simulation through the user interface. these methods call the methods of the Core component with the same name. These methods start/stop the simulation. When the simulation begins, this component also begins to call poll() once in every 10 ms.

## 7.2. Core Component

In this section, a detailed description of the core component will be given. The responsibilities and uses of the core component will be mentioned.

### 7.2.1. Classification

The core component is the module of the system. It will be Java class and there will be only one instance of this class.

### 7.2.2. Definition

The core component is the main module of the system. The purpose of the core component is to construct other components at the beginning of the program and destruct them at end of the program. It will also handle the communication between system components.

### 7.2.3. Responsibilities

The core component will be responsible for the management of the system. It is responsible for construction and destruction of other components. At the beginning of the program the component will construct all other components and at the end of the component the core component will destruct all other components and save necessary information for later use.

The core component will also manage the communications between system components. When a component needs to communicate with another component, it will call the related method of the core component.

### 7.2.4. Constraints

The core component will manage the communication between system components hence it should provide interfaces for each component in the system. Since there can be more than one communication at the same time, the core component should handle them with threads.

### 7.2.5. Composition

The core component does not have any sub component, but it is the manager of all of the system system components.

### 7.2.6. Uses/Interactions

The core component will be in interaction with all components of the system. But the main user of the core component is UI component. When a component needs to communicate with another component, it should call the provided method for that component.

### 7.2.7. Resources

The core component will use built-in thread library of the Java programming languages. Since more than one communication requests can occur at the same time, the core component will start a new thread for each request.

### 7.2.8. Processing

At the beginning of the program the core component will construct VirtualSim component, realSim component, database component, report component and UI component in this order. Then it will wait for communication requests from other components.

When it receives a request, it will start a new thread and in this thread the core component will call related method of the target component. Then it will return the data which is also returned from target component to caller component.

At the end of the program the core component will destruct all the components it constructed and save necessary information to a file for later use.

#### 7.2.9. Interfaces/Exports

The core component will store current session id of the system. And it will also hold references to the constructed components of the system.

## 7.3. VirtualSim Component

In this section, a detailed description of virtualSim component will be given. The work flow of virtualSim component and the input component which is a sub component of virtualSim component will be explained.

#### 7.3.1. Classification

VirtualSim component is a module of the system. It will be a Java class and will contain a sub module named Input component. It will also use trackGenerator component for path generation. There will be only one instance of this component.

#### 7.3.2. Definition

There will be virtual simulation mode in our software which allows users to place virtual sensors on map and make simulation using these virtual sensors. The VirtualSim component will be used for generating predicted path based on joystick input and virtual sensors.

#### 7.3.3. Responsibilities

The virtualSim component will be responsible for simulating target path relying on the virtual sensors. It will also handle the joystick devices via input component. It will use the data that comes from input device and use that data to predict target path. Whenever it receives a poll request from core component, it will return a point array that defines the generated path so far.

The virtualSim component will also provide an interface for listing connected devices and selecting one of them. In addition, the user will be able to define the count and location of the virtual sensors. All of these selections should be made before starting simulation.

#### 7.3.4. Constraints

The virtualSim component will need some configuration before starting simulation. The input device which will be used to determine real path should be set. Also, the count of the virtual sensors and location information for each of them should be given. The UI component will check whether the requirements for input component is satisfied.

The virtualSim component will generate a new position information which belongs to predicted path whenever it receives a poll request from core component. Since the invocation of the poll method will be very frequent, the virtualSim component should be able to return requested result with a very low latency. This causes a necessity to optimize the virtualSim component's poll method and it's sub components' as well.

### 7.3.5. Composition

The virtualSim component has a sub component which is named input component. The detailed design specification of this sub component will be provided later.

### 7.3.6. Uses/Interactions

The virtualSim component will be reachable from core component. The virtualSim component will start simulation when the core component invoke it's startSim method and it will stop simulation when core component invoke it's stopSim method. During simulation, whenever the poll method of the virtualSim component is invoked, it will return an array of points which defines the generated path. The core component will also be able to get real path which is given by joystick device by invoking the getRealPath method of the virtualSim component.

At the inner side, the virtualSim component will be in interaction with input component, trackGenerator component and virtual sensor component. When a poll request is received, the virtualSim component will call input component's poll method and use the returned value to virtual sensors to update their alarm levels. After that it will pass these virtual sensors to trackGenerator component to generate a predicted path.

### 7.3.7. Processing

The virtualSim component will be constructed by core component at the beginning of the program and then it will wait for core component set virtual sensors location and choose active device. After these configurations are made , it will wait for it's startSim method to be called.

The startSim method of the virtualSim component will call startListening method of the input component, initialize the trackGenerator components, set the processing mode to simulation mode and wait for poll and stopSim methods to be invoked. When stopSim method is invoked, the virtualSim component will call stopListening method of the input component and set the component mode to the idle mode.

The poll method is the most important method of the virtualSim component. When it is called, the virtualSim component will call the poll method of input component and retrieve the changes in x and y positions. Then the component will compute this data to determine the current location of the

target and pass this result to virtual sensors to update their alarm level. After that the virtualSim component will send these sensors to trackGenerator component's generate method and return a point array which is returned from trackGenerator component.

### 7.3.8. Interface/Exports

The virtualSim component will store sensor array and real path for the current simulation. It will also holds references to input component and trackGenerator component.

## 7.4. Input Component

In this section we will give detailed a description for input component. We will mention hardware and software constraints of this component. We will also explain the algorithm used by this component and we will refer interactions between input component and other components.

The input component will be used for reading movement data from a joystick or a joystick-like device. Hence the component should manage communication between hardware device (joystick) and related system components. This situation brings hardware constraints and necessity for an algorithm that converts the input data into suitable form for other components. Below we will concern these issues separately.

### 7.4.1. Classification

The input component is a sub module of the virtualSim component. It will be a Java class.

### 7.4.2. Definition

Our software will be able to simulate real path and generated path according to data comes from joystick device. The component that will manage joystick input will be the input component. The purpose of this component is to handle joystick device and forward the data comes from this to related components in suitable form.

### 7.4.3. Responsibilities

The input component will listen for a joystick or a joystick-like device to be connected to the system or to be disconnected from system. When a device is connected or disconnected, the component should notify the user interface component.

The input component will start processing or stop processing when it is notified by user interface component and it will return a response about its current situation.

The main responsibility of the input component is to read raw data from joystick device, convert it into suitable form and forward it to related components. In our case raw data refers to direction

information that comes from joystick device and suitable form refers to x and y position that is computed from direction information. In the computation process, no filtering algorithm will be used but since this process will be real-time, the component might do some filtering to reduce delays.

### 7.4.4. Constraints

The input component will be constructed as our software begins to run. There are two states of this component: Idle state and listening state. Initially the component will be in idle state and will listen for a joystick device that may be connected to the system. When the component receives a request from user interface component, it will switch to listening state and start listening input data from joystick device.

As we have mentioned before, one of the responsibilities of the input component is to handle joystick devices that may be connected to the system. This responsibility causes some hardware constraints we have to concern about. Since there are wide variations of joystick types and brands, it is unexpected for us to support all of them. Our software will only support Logitech joysticks and will provide a configuration interface for other devices. With the configuration interface provided, the users will be able to assign movement actions to their devices' keys or buttons.

When the input component is reading input from joystick device, it should compute and forward the input data in real-time. Hence to lower the latency, the code of this class should be optimized highly. In our software we will accept delays under human perception but if delays exceeds this limit, a filtering algorithm to reschedule output data will be used.

### 7.4.5. Composition

The input component does not have a sub component.

### 7.4.6. Uses/Interactions

The input component will be reachable from user interface component and simulation component. The user interface component will communicate with input component for requesting it to start or stop listening from input device. In addition, the input component will notify the user interface component whether a joystick device is connected to the system. There will be a communication between input component and simulation component when the system is in generating path from virtual sensors mode. The input component will call related method of the simulation component between constant time intervals.

### 7.4.7. Resources

The input component will use Jinput library to handle joystick device. This is a BSD licensed open source library that makes handling joystick devices simpler.

### 7.4.8. Processing

In this section we will explain how the input component perform its duties. We will describe the processes for each state of the component.

At the beginning of our software and after receiving stop listening request from user interface component, the input component will be in idle state. In idle state the only responsibility of the input component is to check whether a device is connected or not. The component will make this check in every 100ms. When a suitable device is detected, it will notify user interface component about the device.

The component will be listening state when it receives a start listening request from user interface component. In listening state, the component will listen for input data that comes from device. For each data read, the component will compute the data and get new x and y position of the virtual target. The computation algorithm is simple: The component will fetch the direction info from received data and update previous x and y position of the virtual target. Then it will append the new position info to the output buffer.

### 7.4.9. Interfaces/Exports

The input component will hold an array of points which refers to changes in x and y axis. It will also store the name of the active device that will be used to gather direction information.

## 7.5. TrackGenerator Component

In this section a detailed description of the trackGenerator component will be provided. The importance and responsibility of the component will be mentioned. The algorithm which is one of the fundamentals of our software and used in this component will explained.

### 7.5.1. Classification

The trackGenerator component is a module of the system. It will be a Java class and will be used by realSim component and virtualSim component hence unlike other components there will be more than one instance of this class.

### 7.5.2. Definition

The trackGenerator component will be used by simulator components which are realSim and virtualSim to generate a predicted path based on sensors' alarm levels. The purpose of this component is to generate a smooth path which is closer to the target's real path.

### 7.5.3. Responsibilities

The trackGenerator component will return a path which is defined by an array of points when requested. The path that is returned should be smooth, realistic and should look similar to target's

real path. To achieve this duty, the trackGenerator component should examine the sensors' alarm levels, locate the target and apply kalman filter to give smoothness to the path.

### 7.5.4. Constraints

The trackGenerator component will be expected to work real-time, hence when requested it should return the generated path immediately. This brings the necessity to optimization of the algorithm code. The main goals of the algorithm is generate a path similar to the real one and do it in very short time interval.

The count of the sensors are also should be limited. Since a large number of sensors can slow down the program and cause unacceptable delays or cause buffer overflow errors, the maximum count of the sensors will be limited to number which will be decided later in develop time.

### 7.5.5. Composition

The trackGenerator component does not have any sub component.

### 7.5.6. Uses/Interactions

The trackGenerator component will be used by two simulator components of the system which are virtualSim and realSim. The component's will provide a method that accepts sensor information and will return an array of points that defines the generated path. The generate method of the trackGenerator component will be called frequently.

### 7.5.7. Resources

The trackGenerator component will use kalman filter when generating track. All points of the path will be hold from the beginning of the simulation to the end of the simulation and kalman filter algorithm will use this array to produce a smooth track.

### 7.5.8. Processing

The trackGenerator component will start processing when it's generate method is invoked. The generate method will accept Sensor class hence the trackGenerator component will retrieve the locations and alarm levels of the given sensors and use them to find new location of the target. The estimated location will be detected by taking weighted average of the sensors' alarm levels. After that the trackGenerator component will add this new position to point array and apply kalman filter to it. Then the component will return the array of points which defines the generated path.

### 7.5.9. Interfaces/Export

The trackGenerator component will hold an array of points which refers to generated track. Since kalman filter needs previous locations, all points that is predicted before will be kept until the end of the simulation.

## 7.6. Logger Component

In this section a detailed description of the logger component will be given. The usage of the logger component will be explained.

### 7.6.1. Classification

The logger component is a module of the system. It will be a static Java class and there will be only one instance of this class.

### 7.6.2. Definition

The purpose of the logger component is to log every action and error that occurred in the system. It will be used by every component in the system.

### 7.6.3. Responsibilities

The logger component is responsible for logging every action and error with detailed information and current date. It should log normal actions into the current session's log file and errors to the error log file.

### 7.6.4. Constraints

The only constraint for logger component is having read/write permissions. The logger component should have permissions to create a file and write into it.

### 7.6.5. Composition

The logger component does not have any sub component.

### 7.6.6. Uses/Interactions

The logger component will provide an interface for other components to log their actions and errors. The other components should provide current session id, their name and log message in order to log their action. The actions will be logged in current session's log file and the errors will be log in general error log file.

### 7.6.7. Processing

The logger component will act differently for each type of logging request. When log method of the logger component is invoked, the component will open the the current session's log file and log the given message with current date and time information. If the file does not exist it will create a new log file for current session.

When the logError method of the logger component is invoked it will open error log file and write the error message with current date, time and session information.

### 7.6.8. Interfaces/Exports

The logger component will hold the directory path of log files. The action and error log files' directories will be different.

## 7.7. Sensor Component

### 7.7.1 Classification

Sensor Component is a module of the system.

### 7.7.2 Definition

The purpose of this componet is to find the position of the target within an area. If system is in 'real simulation mode', we are getting this information from real sensors. If system is in 'virtual simulation mode', we are getting this information from virtual sensors.

### 7.7.3 Responsibilities

We have two types of sensors in our system: a real sensor and a virtual sensor. According to the mode of the system, Sensor Component uses one of the sensor type. Sensor Component communicates with the system and gives information about target's position in constant time interval. This time interval is 10 ms for our system.

All sensors are responsible from setting their alarm levels. This alarm level shows the distance between the sensor and target's location at a specific time.

Moreover, system should know the location of sensors. Real sensors are deployed on a field before the simulation starts. Every real sensor holds its location. If we are in 'virtual simulation mode', virtual sensors are put on a map.

### 7.7.4 Constraints

Only the battery level of the real sensor is important for this component. If the battery is empty, a real sensor can't operate. Except this, this component is completely dependent on the other modules .

It just detects the vibrations and set its alarm level. The states of the system is not important for a real sensor.

### 7.7.5 Compositions

This component is composed of the two independent modules namely real sensor and virtual sensor.

### 7.7.6 Uses and Interactions

Real sensor module has interaction with communication module. Communication module listens to every sensor on the system in a constant time interval. Then communication module sends the data which comes from real sensors to RealSim component. When system listens to a sensor, it tries to get these information:

Sensor id: it is unique for every sensor. It is an integer.

Location: Location of the real sensor on the field. It is a point type.

Alarm level: Alarm level of the real sensor according to the target's position. It is an integer.

Range: It is the maximum distance in which this sensor detects if a vibration occurs. It is a float.

### 7.7.7 Resources

To get information from a real sensor, we need an antenna connected to the sensor. We use Waspmote ZigBee Antenna which is compatible with our sensors. In order to set the location information, a real sensor neeeds GPS. We have a model of A1084 (Vincotech) GPS connected to sensors. Moreover, a real sensor needs a battery as a power supply.

### 7.7.8. Processing

According to the mode of the system, sensor component uses one of the sensor type. In 'real simulation mode', it uses real sensors, in 'virtual simulation mode' it uses virtual sensors. Sensor Component gives information about target's position to the system in constant time interval. This time interval is 10 ms for our system.

Alarm level of a sensor shows the distance between the sensor and target's location at a specific time. All sensors are responsible from setting their alarm levels on this time interval. According to the mode of the system, RealSim or VirtualSim Component gets this data in every 10 ms.

Real sensors get the location information from GPS. Every sensor holds its location information ie. If we are in 'virtual simulation mode', virtual sensors are put on a map. Their location information is got by the system in same way.

### 7.7.9. Interface / Exports

Sensor component has interface with communication, RealSim and VirtualSim components. Thanks to the functionalities of this interface, system can know where the target is at a specific time.

## 7.8. RealSim Component

### 7.8.1 Classification

RealSim component is a module of the system.

### 7.8.2 Definition

The purpose of this componet is to to simulate the target tracking algorithm by using real seismic sensors. If system is in 'real simulation mode', system will simulate tracking of a moving target within an area by using this component.

### 7.8.3 Responsibilities

RealSim Component is designed to simulate the target tracking algorithm by using real seismic sensors. It gets the states of real sensors from communication component in every 10ms. In 'real simulation mode', it starts to listen all sensor's state. RealSim component has a reference of TrackGenerator component. After getting the states of the sensors, it sends them to the TrackGenerator. TrackGenerator generates the new position and path of the target. RealSim Component can send the generated track to UI or Database component of the system. RealSim component is responsible from starting the simulation. It means that communication component starts listening to sensors and TrackGenarator starts to calculating the route of the target when **startSim()** function of RealSim is called.

RealSim component is also responsible from stopping the simulation. It means that communication component stops listening to sensors and TrackGenarator stops calculating the route of the target when **stopSim()** function of RealSim is called.

### 7.8.4 Constraints

There is one constraint for this module. In 'real simulation mode', we will use many sensors in order to calculate the target's position correctly. In this mode, communication component will get alarm level of every sensor in a constant time interval. Since, it is a real time simulation, every component has to do its duties on time. Thus, we should set time interval according to number of real sensors so that we can collect data every of them on time.

### 7.8.5 Compositions

This component is composed of the three independent modules namely Communication component, real sensors and TrackGenerator.

### 7.8.6. Uses and Interactions

RealSim module has interaction with communication module. It gets the states of every sensor on the system in a constant time interval. Then communication module sends the data which comes from real sensors to RealSim component.

RealSim module has interaction with TrackGenerator module. It sends the states of the sensors to TrackGenerator and gets the new position and path of the target from TrackGenerator.

Moreover, RealSim module has interaction with Core Component so that it can send the generated track to UI or Database component of the system.

### 7.8.7. Resources

We have Waspmote IDE to write and compile our codes when we will use Waspmote Seismic Sensor Card and Waspmote Gateway. Waspmote Gateway is a hardware device which composes communication component. To listen the usb port on which Waspmote Gateway is connected, we use Waspmote IDE. Also when we want to upload our codes to Waspmote Seismic Sensor Card, we need this software.

### 7.8.8. Processing

When the system is 'real simulation mode', **startSim()** function of RealSim is called. Then, RealSim component starts the simulation. In other words, communication component starts listening to sensors and TrackGenarator starts to calculating the route of the target.

Sensor component detects target's position and sets its alarm level according to the distance between itself and the target. RealSim component calls the **poll()** method of itself in every 10ms. So, communication componnet gets the alarm level of all sensors in this time interval. When user wants to stop the 'real simulation mode', **stopSim()** function of RealSim is called. Then, RealSim component stops the simulation. In other words, communication component stops listening to sensors and TrackGenarator stops to calculating the route of the target.

### 7.8.9. Interface / Exports

RealSim Component has interface with Communication Component so that it can listen to real sensors. RealSim Component has interface with TrackGenerator Component so that it can get the path of the target calculated.

RealSim Component has interface with Core component so that it can send the generated track to UI or Database component. RealSim Component can start and stop simulation owing to the functionalities provided by this interface.

## 7.9. Communication Component

### 7.9.1. Classification

Communication component is a sub module of the system.

### 7.9.2. Definition

Communication component is a hardware device that communicates with real seismic sensors and RealSim Component. It is connected to the system with usb port.

### 7.9.3. Responsibilities

When system is in 'real simulation mode', communication component listens to the real sensors. It gets the alarm levels of the sensors in every 10 ms. It also holds the location, sensorId and states of every sensor in its memory. When this data is needed, RealSim component gets and uses.

### 7.9.4. Constraints

There is one data type for the Communication module. It is **realSensor** array. In this data type, it holds the location, sensorId and alarm levels of every sensor. To operate properly every sensor has to be introduced to the system with a unique id. Also, sensors have to use the same data type with communication component so that it can get and interpret the raw data to a meaningful data.

### 7.9.5. Compositions

Since there is only one component, it is the main component of the sub module.

### 7.9.6 Uses and Interactions

Communication module has interaction with RealSim module. It gets the states of every sensor on the system in a constant time interval. Then communication module sends the data which comes from real sensors to RealSim component.

### 7.9.7 Resources

Waspmote IDE is needed to listen the usb port on which Waspmote Gateway is connected. Moreover, compatible driver for this device has to be installed to the computer.

### 7.9.8. Processing

System will have many seismic sensors deployed on field. In 'real simulation mode', they will be operating. They detect any vibration on that area and generate alarm signal showing the strength of the vibration. At this point, system needs a component to get these alarm levels of the sensors in a constant time intervals. Communication Component of our system, gets these alarm levels from sensors.

### 7.9.9. Interface / Exports

Comunication Component has interface with RealSim and Sensor Components. Owing to the functionality provided by this interface, communication component gets the states of the sensors.

Moreover, it can send the states of the real sensors and their location information to RealSim Component.

## 7.10 Database Component

### 7.10.1 Classification

The Database component is one of the main modules of the Target Tracking system.

### 7.10.2 Definition

The Database component's main purpose is to organize the database of the Target Tracking system in order to provide an efficient way of accessing and storing the data items of the application.

### 7.10.3 Responsibilities

The Database component is responsible of performing the operations of storing and retrieving information from the database.

The Core component will ask the Database component to store any new or updated data entity entries, such as Session, Sensor, Real Track or Generated Track and the Database component is responsible of performing the corresponding update queries to on the database.

The Report Generator component will ask the Database to retrieve information related to a specific time interval together with all the corresponding track generation information, such as session Ids, real tracks (if in VirtualSim mode), generated Tracks for each real track, used sensors and their locations. The Database component is responsible for performing the corresponding query and returning the result Set containing this information.

### 7.10.4 Constraints

There are some restrictions/limitations for Database component. PostgreSQL will be used as the object-relational database system since it allows storage of position items as longitude and latitude and it also strongly conforms to the ANSI-SQL:2008 standard.

The most frequent storage operations are modeled as specific methods for efficiency purposes. The information retrieving methods used by the Report Generator component use some expensive join operations of all the related tables using their common key, sessionID. This efficiency issue needs to be addressed by performing an optimization on the query corresponding to this operation.

### 7.10.5 Compositions

This component has no subordinates.

### 7.10.6 Uses and Interactions

The Database component is used by the Report Generator component and the Core component.

The Core component uses the Database component for storing information about any change made by the user through the UI component, including adding/removing/relocating a sensor, starting/ending the generation of a track in both virtual and real modes.

The Report component is the source of the need of the Database component. The Tracking System provides the user the functionality of generating a report of the tracking made within a time interval. Hence all the information related to the tracking made within the interval should be kept organized in the database, which will be controlled and accessed through the Database component. The Report Component uses the Database component for retrieving the information necessary for the report.

### 7.10.7 Resources

The resource used by the Database component, as the name implies, is the database of the System. The database is composed of the Sensor, Alarm, RealTrack,GeneratedTrack, TrackLocation, SensorLocation entities.

The Database component stores information in each of these entities at the end of a specific session. All the added/changed data (sensors, tracks) is then reflected to the database at the end of a session, together with an automatically generated unique id for a session, the SessionID.

The Database component retrieves information belonging to a session or more, within a time interval, by selecting the desired information from the Sensor, RealTrack, GeneratedTrack tables according to their common sessionID.

### 7.10.8. Processing

This section will describe how the information storing and retrieving operations are performed.

The store operations (storeSessionInfoUpdate(), storeSensorInfoUpdate(), storeRealTrackUpdate(), storeGeneratedTrackUpdate()) all use a standard query of the form

```
INSERT INTO tableName [ ( fieldName,...  ) ]
{ VALUES ( {fieldValue, ...}) }
```

The tableName, fieldName, and fieldValue are subject to change according to the method.

The information retrieving operations( executeQuery()) uses a standard query of the form

```
SELECT * FROM tableName1, tableName2,... WHERE
tableName1.sessionID = tableName2.sessionID.id AND
tableName2.sessionId =...;
```

### 7.10.9. Interface / Exports

The interface of the Database component is described as follows:

storeSessionInfoUpdate( sessionID : int,startTime : date, endTime : date) : int

This method takes as argument the sessionID, the start and end time of the session this sessionID describes. It then performs a INSERT query as described in secion 7.10.8, and inserts a new row in the Session entity.

storeSensorInfoUpdate( sessionID : int, Sensor[] sensors) : int

This method takes as argument the sessionID, a Sensor array containing the sensors and their related information. It then performs a INSERT query as described in secion 7.10.8, and inserts a new row in the Sensor and corresponding SensorLocation entity.

storeRealPathUpdate( sessionID : int, Point[]) : int

This method takes as argument the sessionID, a Point array containing the points that compose the real path received by the input device. It then performs a INSERT query as described in secion 7.10.8, and inserts a new row in the RealTrack entity and the corresponding TrackLocation entity.

storeGeneratedPathUpdate (sessionID : int, Point[]) :int

This method takes as argument the sessionID, a Point array containing the points that compose the path generated by our Tracking system. It then performs a INSERT query as described in secion 7.10.8, and inserts a new row in the GeneratedTrack entity and the corresponding TrackLocation entity.

createQueryStatement(fieldName : string, fieldValue : string, tableName : string) : string

The method takes as arguments the fieldName and the corresponding fieldValue to be selected. It forms a string that represents a simple select query and returns it.

createUpdateQuery(updateCommand : string, fieldName : string, fieldValue : string, tableName : string ) : string

The method takes as arguments the fieldName and the corresponding fieldValue to be updated/deleted/inserted. It forms a string that represents a simple update query and returns it.

executeQuery(string query):resultSet

This method executes the query perviously formed by the createQueryStatement() method.

executeUpdateQuery(query) : int

This method executes the update query returned by the createUpdateQuery() method.

## 7.11 Report Generator Component

### 7.11.1 Classification

The Report Generator component is one of the main modules of the Target Tracking system.

### 7.11.2 Definition

The Report Generator component is the component that provides the user of the Target Tracking System with the functionality of generating a report containing all the relevant tracking information done within a time interval.

### 7.11.3 Responsibilities

The Database component is responsible of the creation of a report document showing graphically the tracked targets and the used sensors within a time interval.

The UI component will ask the Report Generator component to generate a report in a specified format and user-specified start and end time of the interval.

Then the Report Generator component will ask the Database to retrieve information related to a specific time interval together with all the corresponding track generation information, such as session Ids, real tracks (if in VirtualSim mode), generated Tracks for each real track, used sensors and their locations. The Database component is responsible for performing the corresponding query and returning the result Set containing this information.

The Report Generator will then parse the resultSet returned by the generateReport() method and create an XML file out of it. The PDF/doc document will be eventually prepared by parsing this XML file. The Report Generator component will store the XML file in case the use decides to store the generated document.

### 7.11.4 Constraints

There are restrictions/limitations for Report Generator component. The format of the report can either be PDF or DOC. The time interval selected should not exceed one day time interval and should not be shorter than a session time (the time for a track to be started, ended, and estimated).

### 7.11.5 Compositions

This component has no subordinates.

### 7.11.6 Uses and Interactions

The Report Generator component is used by the UI component.

The Report Generator component uses the Database component eventually to perform the task of report generating.

The Report component is the source of the need of the Database component. The Tracking System provides the user the functionality of generating a report of the tracking made within a time interval. Hence all the information related to the tracking made within the interval should be kept organized in the database, which will be controlled and accessed through the Database component. The Report Component uses the Database component for retrieving the information necessary for the report.

### 7.11.7 Resources

The resource used by the Report Generator component, as we previously stated that the Report Generator will primarily use the Database component, is the database of the System. The database is composed of the Sensor, Alarm, RealTrack,GeneratedTrack, TrackLocation, SensorLocation entities.

The Report Generator component retrieves with the means of the Database component information belonging to a session or more, within a time interval, by selecting the desired information from the Sensor, RealTrack, GeneratedTrack tables according to their common sessionID to generate a graphical representation of them in the form of PDF or doc document.

### 7.11.8. Processing

Since the only method of the Report Generator component, besides the getters and setters of its attribute, is the generateReportMethod(), we will focus mainly in this method's processing.

This is an information retrieving operation. The start time and end time of the interval the user wants to generate the report for, specify two important arguments for a query which needs to be performed by the database.

The query should return the sessionIDs of the session whose start time corresponds to the start time of the interval of the report, the session whose end time corresponds to the end time of the interval of the report, and all the sessions in between.

This information retrieving operations( executeQuery()) needs the composition of a query of the form

```
SELECT sessionID FROM Session WHERE startTime >= argStartTime OR endTime <= argEndTime
```

The query is formed and then executed using the Database component's corresponding methods.

### 7.11.9. Interface / Exports

The interface of the Database component is composed of the getters and setters of the attributes and a generateReport() method.

getIntervalStartTime() : date

Gets the start time of the interval the report is being generated for.

setIntervalStartTime(startTime : date) : void

Sets the start time of the interval the report is being generated for.

getIntervalEndTime() : date

Gets the end time of the interval the report is being generated for.

setIntervalEndTime(endTime : date) : void

Sets the start time of the interval the report is being generated for.

getReportFormat() : int

Gets the format of the document of the report.

setReportFormat(format : int) : void

Sets the format of the document of the report.

+generateReport(startTime : date, endTime : date, format : int) : file

This method takes as argument the start and end time together with the format of the document to be generated. It then calls the Database component's createQueryStatement() according to these arguments, and then the executeQuery() method with an argument the query returned by the previous method. Then the returned resultSet is parsed and stored into an XML file.

+saveGeneratedReport(filePath : string, fileName : string)

This method takes as argument the fileName and the filePath as the name and location of the generated document to be saved.

## 8. Libraries and Tools

**PostgreSQL[3]:** Because it is an object-relational database managment system (ORDBMS) available for many platforms including Linux, FreeBSD, Solaris, MS Windows and Mac OS X.

**Eclipse RCP[2]:** It is an IDE that we are going to use for design. It is very compatible with Java programming language, and with the library provided by Aselsan.

**Java Swing:** It's a library to design frames, windows in given environment. In other words, it is a library to design GUI.

**Waspmote Seismic Sensor Card, Waspmote Gateway[1]:** These hardware devices have been provided by ASELSAN.

**Waspmote IDE[1]:** We are using the Waspmote IDE to write and compile our codes when we will use Waspmote Seismic Sensor Card, Waspmote Gateway.

We are also provided some libraries like lowagie-itext, sun-pdfview, jinput-windows libraries.

## 8. Time Planning

The Gannt chart for both terms is below.

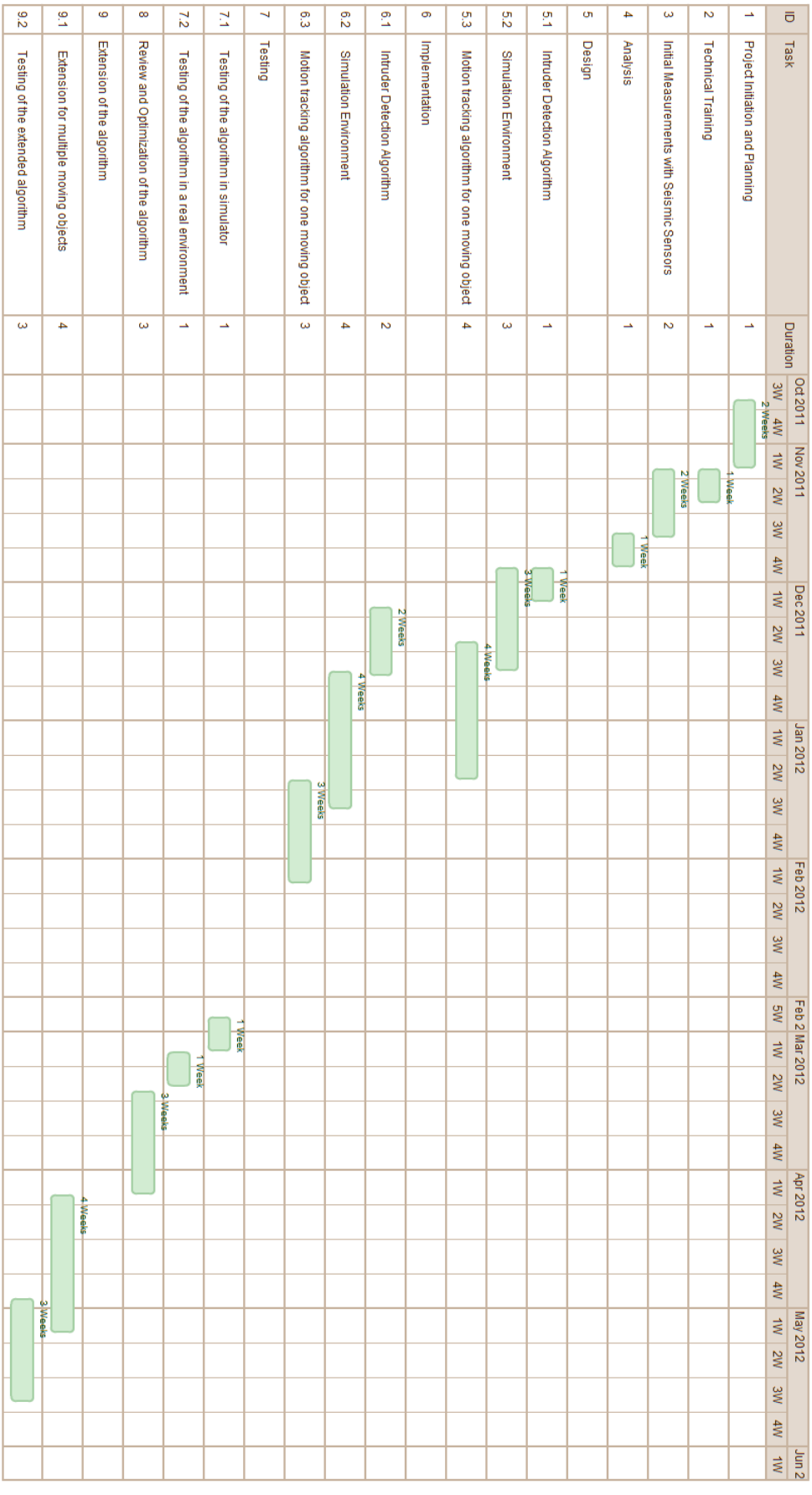



Figure 40: Gannt Chart

# 9. Conclusion

In this document, we have talked about the initial design specifications of our project. We have discussed design considerations, data design, system architecture, libraries and tools.

After the completion of this project, the end product will be used by Turkish Company ASELSAN. Possible scenarios for the usage of this product are border control, battle field surveillance, traffic flow measuring, or animal monitoring, etc.