DETAILED DESIGN REPORT

for

CONTEXT AWARE USER INTERFACE PROJECT

MOMO SOFTWARE

Burak Kerim Akkuş - 1559855

Ender Bulut - 1559996

Hüseyin Can Doğan - 1560077

1 INTRODUCTION

This detailed design report is prepared by Momo Software to show the progress we have shown in this project. The report includes initial descriptions of the proposed software system design. Additionally, it also includes current project status and schedule.

1.1 Problem Definition

The problem definition of the project had been described in Software Requirement Specification Report. Mobile Device technology has an important role in our lives in many areas such as in daily lives, in military services etc. As the usage of mobile devices increases by day by, the expectations of users also increase. The most important one of these expectations is easy and effortless usage for mobile device users in various areas. However, users have difficulties during changing environment conditions. For example, you may not use these small devices easily and without making effort for different contexts, different light conditions, different colored environments or movement. CAUI will be a very efficient answer for this problem by adapting itself in changing environment conditions.

1.2 Purpose

The purpose of this document is to describe the detailed design process about the project. The requirements specified in the requirements specification report will be explained in detail as the structural components which will be used in the implementation phase in this document. We can determine the design issues such that system architecture, data design, user interface.

1.3 Scope

The scope of Context Aware User Interface had been stated substantially in requirement specification report. In this section, we will state what it has been written in our previous report in summary. Then, we will provide a future insight for the design scope of Context Aware User Interface.

Our project provides different types of communication models as one to one and many to many. That is, a mobile device in Sender mode can send data to a unique mobile device in Receiver mode or many devices all in Receiver mode. Moreover, users can receive missions from Center.

Our project will support data types for our communication model. One of them will be continuous communication including map info, environment info and feedback and GPS info. The other one will be single communication containing messages and tasks.

1.4 Overview

This document has six additional chapters. The second chapter (System Overview) is designed to reflect a general description of the software system including its functionality and matters related to the overall system and its design. The next section is Design Considerations including special design issues that need to be addressed or resolved before attempting to devise a complete design solution. The fourth chapter is about data description and data dictionary. The fifth chapter (System Architecture) is prepared to show a description of the program architecture. The User Interface Design chapter includes overview of user interface, screen images, screen objects and actions. The next one about detailed design contains the internal details of each design component. The next chapter is to represent which libraries and tools will be used in the project. The last section is time planning to be able to plan the team structure, estimation (basic schedule) and process model for first and second terms.

1.5 Definitions, Acronyms and Abbreviations

SDD: Software Design Document
PDA: Personal Digital Assistant
GUI: Graphical User Interface
3G: 3rd Generation
API: Application Programming Interface
CAUI: Context Aware User Interface
MDA: Mobile Device Application
MDSC: Mobile Device System Controller
MS: Main Server
WNC: Wireless Network Connection
I/O: Input/output

1.6 References

IEEE Recommended Practice for Software Design Descriptions (IEEE Std 1016-1998).

Çakıcı, Ruken "Software Design Document Template"
<https://cow.ceng.metu.edu.tr/Courses/?semester=20101&course=ceng491&cedit=0>.

2. SYSTEM OVERVIEW

2.1 Functionality

The basic functionality of our project is related with the idea, namely "Context Aware User

Interface". It is called as context aware, because its structure will be changed according to the environment and user conditions; for example lighting and acceleration of the user.
In order to give meaning to this interface, we will create a scenario which is another important functionality of our system. Our scenario is basically represents a military environment which are created by mobile device application, mobile device sensor controller and information server application. These units enable us to create a war environment that includes mobile device user and its interactions between information server application (sending messages to other units, viewing and analyzing war environment etc.) By using this scenario we can apply "context aware user interface" idea and create our project by combining these two main functionalities.

Context aware concept requires a user interface whose content and structure can be changed according to the environment conditions automatically. For example when a user in the environment where the lighting density is very low, he/she still should be able to view the mobile device screen and analyze the war environment. In other words user interface structure should be changed in order to remove the negative effect of environment and help the user. This functionality is same when we consider the user's acceleration. Our interface should be changed when the user moves by changing the font of the letters or zooming in the most critical parts of UI.

2.2. Idea of Project

The idea that is behind our project is basically context related problems for UI's. They are influenced very easily by the environment conditions. The small changes of lighting density or user condition directly affect the performance and usability of user interface. In order to prevent this negative effect, we create a user interface that is aware of environment context.

2.3. Audience and Benefits

We define our audience as military forces, because this audience makes our project and idea more meaningful. Actually this idea can be implemented for civilians and our product can be used for daily-life. However we choose military forces in order to reach more critical and beneficial results. In the war environment, time is an important issue and even small amount of times can affect the progress and result of the war. Therefore, our product firstly gives more time to the units by offering CAUI. Secondly they can analyze the information in a more stable and understandable way.

3. DESIGN CONSIDERATIONS

3.1. Design Assumptions, Dependencies and Constraints

3.1.1 Constraints

For designing our system and defining design considerations, we consider a scenario that we can apply CAUI logic. Basically, our scenario depends on a war environment that a user can analyze and view it by using mobile device and our applications. In order to successfully apply our design, we create a system which includes MDA, MDSC and MS. These main parts are mentioned at this point because communication between MDA and MS requires an important constraint that is WNC. We need to use WNC, because it helps to connect and communicate MDA and MS.

Another important constraint is device availability. While creating our software and design our product, we plan to proceed with emulator rather than a real mobile device. First of all our sponsor states that they can not give us a mobile device which means either we should find a mobile device or we should progress with emulator on computer. We choose to use Android based emulator, since we can develop our system more easy and flexible way. Moreover, Android platform is well-defined and can answer our sensor-related issues. As explained in our SRS, environment's light info or user's movement info is very important metrics for CAUI concept. In order to create and change UI, we have to collect this environment info and Android platform answers all of these needs related to context awareness.

3.1.2 Assumptions

When we design our system, we need to make some assumptions that enable us to progress and finish the system successfully. A device simulator and its importance are explained at the previous parts shortly. At this point, we make some assumptions that are related to simulator. First of all, we assume that it has capability that enables us to learn light info and acceleration info by using its light sensor and accelerometer. Second assumption is that the emulator should enable us to apply WNC logic correctly. In other words, we should use WNC for our application on the emulator without facing any problems. GPS and compass support are other assumptions that we make. In order to define the location of the user, we assume we can apply GPS on the emulator and get location of the user. Moreover, we expect that we will use compass support of android platform without facing any problem or difficulties.

3.1.3 Sponsor requests

As we mentioned, we need a scenario in order to implement CAUI concept. Since our sponsor is ASELSAN, we think that it is suitable and acceptable that the scenario is related to military.

Therefore, we plan and develop such a scenario that we can apply a war environment for the system and analyze how our applications work correctly for context awareness.

3.2. Design Goals and Guidelines

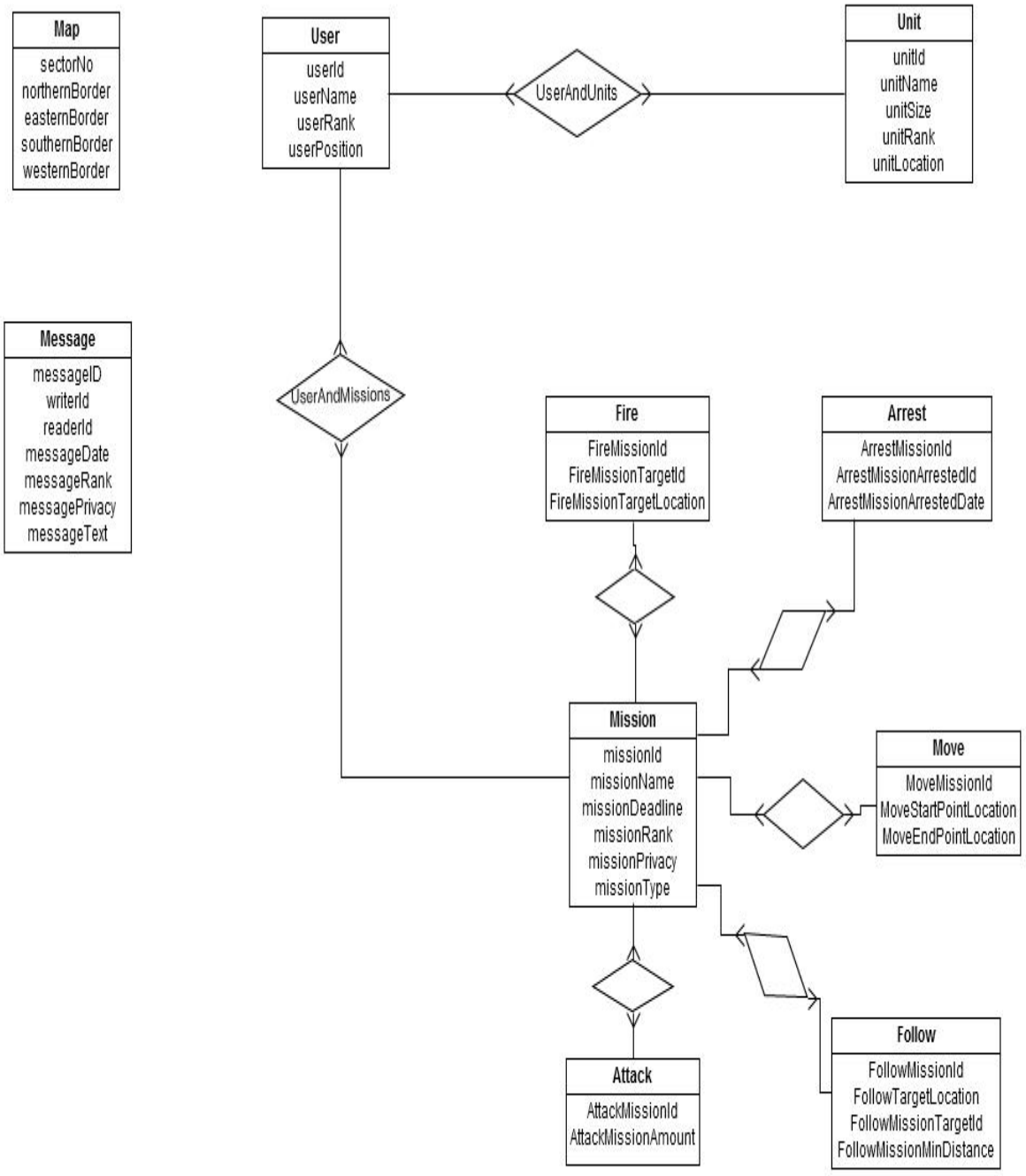
3.2.1 Design priorities

Under design considerations title, we have to talk about the priorities of goals that we want to achieve for the future plans. First of all we have primary design priorities that are related to GUI and context awareness with just file I/O. In other words our first goal is to implement GUI and context awareness issue without using scenario and network. After achieving first goal, the second design priority is to add a scenario and database to system and to associate theses eparateconcepts.

3.2.2 Final Objective

After all of the implementations, our final objective is to create a complete product that can be used by many users simultaneously without facing any memory, speed or other problems by keeping our design as simple as possible. Keeping the design simple is important, since our final product includes many concepts such that; GUI, context awareness WNC, database implementation and networking.

4.1 Data Description



Database and database management system have a high importance for our project. We need to store some data in database in order to process the system.

Firstly, we have a map image in the screen of user's mobile device. The map information has a very important role for our scenario due to requirements of users. All map information of an area should be ready continuously for users. Therefore, all map information should be stored in a storage area. CAUI will be able to use a database that is prepared according to database design specifications that are determined in this section of the report.

It is stated that map information should be stored in database. In detail, map will be separated into sectors and each sector has four borders for four main directions. It will be used to send correct map information according to user's coordinates. In addition to map information, user information should be stored in database. That is, the system need to store what user identification number is and user name is etc.

Another important data is unit information that has to be stored in database. Those information includes identification info and what unit size is, what unit rank is, what unit location is. These are necessary to build up a connection between units or to watch units from center. Initially, it is described how the major data or system entities are stored, processed and organized with tables below. After these initial stored data, it will be stated which information can be written to the database to be able to be read from all users.

4.1.1 User

User table will be used to store information for each user that can be "log-in" state in the system. This component is basically responsible for holding the info of the user. Some applications of our system depends on the user attributes, for example; id number,rank or coordinates of the user. For making these applications without any problems, we use User class.

Data Name	**Data Type**	**Size**	**Range**
userId	Integer	4 Bytes	
userName	String	5-100 Characters	
userRank	Integer	4 Bytes	
userPosition	Coordinate	12 Bytes	(Float altitude, Float latitude, Float longitude)

4.1.2 Unit

Unit table stores specific unit attributes storing basic unit information. These attributes are unitId, unitName, unitSize, unitRank and Location. These information are very important important to be known for military system. The unitId attribute and userId in User table have a realationship, respectively. That is, a user may be part of more than one unit or a unit may consist of many users.

Data Name	Data Type	Size	Range
unitId	Integer	4 Bytes	
unitName	String	2-50 Characters	
unitSize	Integer	4 Bytes	
unitRank	Integer	4 Bytes	
unitLocation	Coordinate	12 Bytes	(Float altitude, Float latitude, Float longitude)

4.1.3 Mission

This table stores predefined missions that can be assigned to a unit, many units, a user and many users that will act as how they are ordered with respect to missions specified by a unique id number, name, deadline, rank and privacy. Another important attribute is mission type. The missionId attribute has a relation with userId in User table.

Data Name	Data Type	Size	Range
missionId	Integer	4 Bytes	
missionName	String	2-500 Characters	
missionDeadline	Date	20 Bytes	(Integer year, Integer month, Integer date, Integer hrs, Integer min)
missionRank	Integer	4 Bytes	
missionPrivacy	Integer	4 Bytes	
missionType	Enum	4 Bytes	ARREST, ATTACK, STOP, MOVE, FIRE, FOLLOW

4.1.4 Arrest

Arrest mission table stores only arrest mission specific parts of missions. Its arrestMissionId attribute is both primary key and foreign key which references missionId attribute of Mission table.

Data Name	Data Type	Size	Range
arrestMissionId	Integer	4 Bytes	
arrestMissionArrestedId	Integer	4 Bytes	
arrestMissionArrestedDate	Date	20 Bytes	(Integer year, Integer month, Integer date, Integer hrs, Integer min)

4.1.5 Attack

Attack mission table stores only attack mission specific parts of missions. Its attackMissionId attribute is both a primary key and a foreign key which references missionId attribute of Mission table.

Data Name	Data Type	Size	Range
attackMissionId	Integer	4 Bytes	
attackMissionAmount	Integer	4 Bytes	

4.1.6 Fire

Fire mission table stores only fire mission specific parts of missions. Its fireMissionId attribute is both a primary key and a foreign key which references missionId attribute of Mission table.

Data Name	Data Type	Size	Range
fireMissionId	Integer	4 Bytes	
fireMissionTargetId	Integer	4 Bytes	
fireMissionTargetLocation	Coordinate	12 Bytes	(Float altitude, Float latitude, Float longitude)

4.1.7 Follow

Follow mission table stores only fire mission specific parts of missions. Its followMissionId attribute is both a primary key and a foreign key which references missionId attribute.

Data Name	Data Type	Size	Range
followMissionId	Integer	4 Bytes	
followTargetLocation	Coordinate	12 Bytes	(Float altitude, Float latitude, Float longitude)
followMissionTargetId	Integer	4 Bytes	
followMissionMinDistance	Float	4 Bytes	

4.1.8 Move

Move mission table stores only move mission specific parts of missions. Its moveMissionId attribute is both a primary key and a foreign key which references missionId attribute of Mission table.

Data Name	Data Type	Size	Range
moveMissionId	Integer	4 Bytes	
moveStartPointLocation	Coordinate	12 Bytes	(Float altitude, Float latitude, Float longitude)
moveEndPointLocation	Coordinate	12 Bytes	(Float altitude, Float latitude, Float longitude)

4.1.9 Map

In Map table, map information will be stored. A map will be seperated into many sectors and each sector has four borders for four main directions. That is, what unique sector number is, what northern border, what southern border is, what eastern border is and what western border is will be stored in database to be able to get suitable map information with respect to coordinates of user. Differentiating attribute of the map table is sectorNo attribute, and so it is the primary key of the table.

Data Name	Data Type	Size	Range
sectorNo	Integer	4 Bytes	
northernBorder	Float	4 Bytes	
easternBorder	Float	4 Bytes	
southernBorder	Float	4 Bytes	
westernBorder	Float	4 Bytes	

4.1.10 Message

In Message table, all messages that provide connection between both users and units in CAUI system will be stored. This information will only include common types of system attributes as designed in system hierarchy previously. This table stores specific attributes of message information such as sender, receiver, data etc. Differentiating attribute of the table is messageId attribute, and so it is the primary key of the table.

Data Name	Data Type	Size	Range
messageID	Integer	4 Bytes	
writerId	Integer	4 Bytes	
readerId	Integer	4 Bytes	
messageDate	Date	20 Bytes	(Integer year, Integer month, Integer date, Integer hrs, Integer min)
messageRank	Integer	4 Bytes	
messagePrivacy	Integer	4 Bytes	
messageText	String	0 – 500 Characters	

.

4.2 Data Dictionary

User

User = userId
+ userName
+ userRank
+ userPosition

userId = *id number of user*
userName = *name of the user*
userRank = *rank of the user*
userPosition = *coordinate values of the user*

User
userID : int userName : string userRank : int userPosition : Coordinate
User (string xmlData) User (string rawData) getUserId () : int getUserName () : string getUserRank () :int getUserPosition () : Coordinate setUserId (int id) : void setUserName (string name) : void setUserRank (int rank) : void

Unit

Unit = unitId
+ unitName
+ unitSize
+ unitRank
+ unitInfo

unitId = *id number of unit*
unitName = *name of the unit*
unitSize = *size of the unit*
unitRank = *rank of the unit*
unitInfo = *info of the unit*

Unit
unitID : int unitName : string unitRank : int unitSize : int unitInfo : string
Unit (string xmlData) Unit (string rawData) getUnitId() : int getUnitName() : string getUnitSize() : int

getUnitRank() : int getUnitInfo() :string setUnitId(int id) : void setUnitName(string name) : void setUnitSize(int size) : void setUnitRank(int rank) : void

Mission

Mission = mission_id
+ mission_name
+ deadline
+ mission_rank

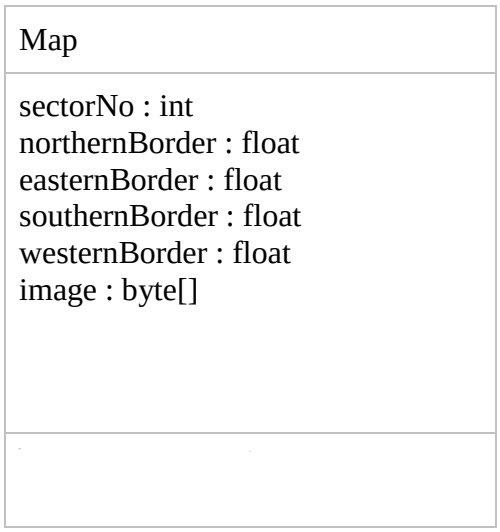
missionId = *id number of specific mission*
missionName = *name of the mission*
missonDeadline = *deadline of the mission*
missionRank = *rank of the mission*
missionPrivacy = *privacy of the mission*

Mission
missionID : int missionName : string missionRank : int missionPrivacy : int missionDeadline : date
Mission (string xmlData) Mission (string rawData) getMission Id () : int getMissionName () : string getMissionRank () :int getMissionPrivacy () : int getMissionDeadline() : date setMissionId (int id) : void setMissionName (string name) : void setMissionRank (int rank) : void setMissionPrivacy (int class) : void setmissionDeadline(date time) : void

Map

Map = sectorNo : int
+northernBorder : float
+ easternBorder : float
+ southernBorder : float
+ westernBorder : float
+ image : byte[]

sectorNo = *it shows sector number*
northernBorder : *northern boundary of the map*
easternBorder : *eastern boundary of the map*
southernBorder : *southern boundary of the map*
westernBorder : *western boundary of the map*
image : byte[] : *byte array that compose the image*



Message

Message = messageID
+ writeID
+ readerID
+ messageDate
+ messageRank
+ messagePrivacy
+ messageText

messageID = *id number of the message*
writerId = *user id of user who sends to the message*
readerId = * user id of user who reads to the message *
messageDate = *date when the message is sent*
messageRank = * importance of the message *

messagePrivacy = *shows privacy of message that defines who has reading right of message*
messageText = *stored message text*

Message
messageID : int writeID : int readerID : int messageDate : date messageRank : int messagePrivacy : int messageText : string
Message (string xmlData) Message (strin rawData) getMessageID() : int getWriteID() : int getReaderID() : int getMessageDate() : date getMessageRank() : int getMessagePrivacy() : int getMessageText() : string setMessageID(int id) : void setWriteID(int id): void setReaderID(int id) : void setMessageDate(date time): void setMessageRank(int rank) : void setMessagePrivacy(int privacy) : void setMessageText(sting text) : void

Coordinate

Coordinate = altitude
+ latitude
+ longitude

altitude = *altitude value of coordinate*
latitude = *latitude value of coordinate*
longitude = *longtitude value of coordinate*

Coordinate
altitude : float latitude : float longitude : float
Coordinate (float alt, float long, float lat) getAltitude() : float getLatiitude() : float getLongiute() : float

MainServerCore

MainServerCore = databaseManager
+ missionManager
+ messageManager
+ mapManager
+ authenticationManager
+ transmissionManager

databaseManager = *DatabaseManager object*
missionManager = *MissionManager object*
messageManager = *MessageManager object*
mapManager = *MapManager object*
authenticationManager = *AuthenticationManager object*
transmissionManager = *TransmissionManager object*

MainServerCore
databaseManager : DatabaseManager missionManager : MissionManager messageManager : MessageManager mapManager : MapManager authenticationManager : AuthenticationManager transmissionManager : TransmissionManager

MissionManager

MissionManager = tmpMissionList

tmpMissionList = *Mission objects' list that hold Missions temporarily*

MissionManager
tmpMissionList : Mission[]
recordMissiontoDB (Mission newMission) : void getMissionsOfUser (int userID) : string

MessageManager

MessageManager = tmpMessageList : Message[]

tmpMessageList = *Message objects' list that hold Messages temporarily*

MessageManager
tmpMessageList : Message[]
recordMessagetoDB (Message newMessage) : void getMesaagestoUser (int UserID) : string getMessagesfromUser (int UserID) : string getMessagestoAll() : string getUnreadMessagestoAll () : string getUnreadMessagestoUser (int userID) : string

MapManager

MapManager
convertCoordinatestoSector(float altitude, float latitude) : int getMap (int sectorNumber) : Map

AuthenticationManager

AuthenticationManager = count

count = *counts the number of wrong login requests*

AuthenticationManager
count : int
authenticationResult(int userID, string pass) : boolean recordLoginHistory() : void

TransmissionManager

TransmissionManager = tmpData
+ dataTypeId

tmpData = *holds the required data temporarily*
dataTypeId = *shows the type of data that is stored*

TransmissionManager
tmpData : string dataTypeId : int
receiveDataFromMobileDevice () : string sendDatatoMobileDevice(int type, string text) : boolean

MobileApplicationCore

MainServerCore = inputController
+ userInterfaceController
+ sensorController
+ userInterfaceSettingsManager
+ mobileDeviceTransmissionUnit

inputController = *InputController object*
userInterfaceController = *UserInterfaceController object*
sensorController = *SensorController object*
userInterfaceSettingsManager = *UserInterfaceSettingManager object*
mobileDeviceTransmissionUnit = *MobileDeviceTransmissionUni object*

MobileApplicationCore
inputController :InputController userInterfaceController :UserInterfaceController sensorController: SensorController userInterfaceSettingsManager: UserInterfaceSettingManager mobileDeviceTransmissionUnit: MobileDeviceTransmissionUnit

InputController

MainServerCore = pixelValueHorizontal
pixelValueVertical

pixelValueHorizontal = *horizontal pixel value of the point that was selected by the user*
pixelValueVertical = *vertical pixel value of the point that was selected by the user*

InputController
pixelValueHorizontal : int pixelValueVertical : int
controlPixelValues(int horizontal, int vertical) : void missionIconChosen : void messageIconChosen : void teamInfoIconChosen : void mapInfoIconChosen : void

MobilDeviceTransmissionUnit

MainServerCore = tmpData
dataTypeId

tmpData = *holds the required data temporarily*
dataTypeId = *shows the type of data that is stored*

MobileDeviceTransmissionUnit
tmpData : string dataTypeId : int
receiveDataFromMainServer () : string sendDatatoMainServer(int type, string text) : boolean

UserInterfaceController

UserInterfaceController
+displayIcons():void +displayMap():void +displayMissionList():void +displayCompletedMissions():void +displayIncompletedMissions():void +displaySentMessages():void +displayReceivedMessages():void +displayTeamInfo():void +displayMapInfo():void

UserInterfaceSettingsManager

UserInterfaceSettingsManager = iconWidth
+ iconHeight

UserInterfaceSettingsManager
+iconWidth:int +iconHeight:int
+resizeIcons(int width, int height) : void +resizeMap(int width, int height) : void +changeIconsOrder(): void

SensorController

MainServerCore = brightnessValue
+ accelerationValue
+ directionValue
+ longtitudeValue
+ lattitudeValue

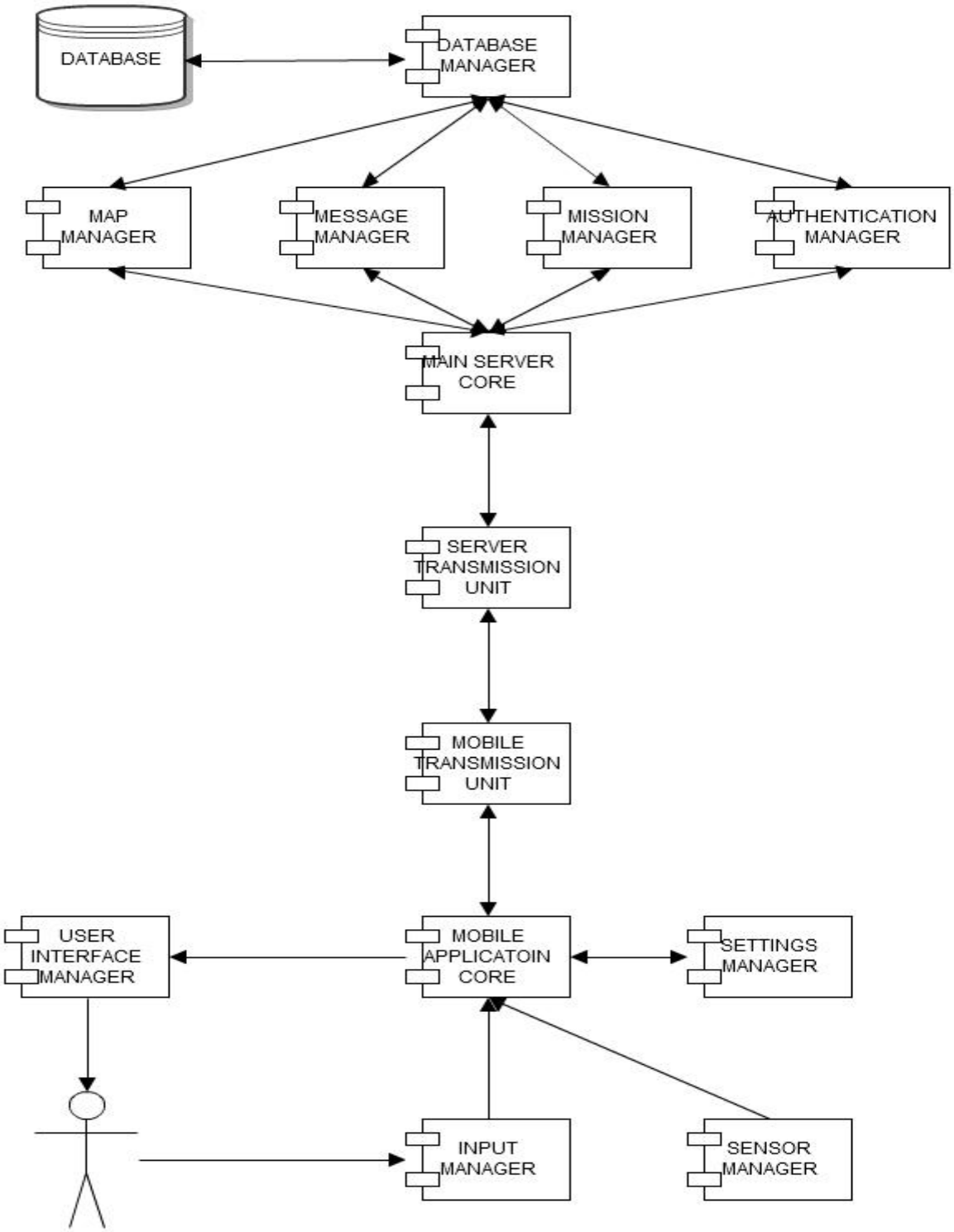
brightnessValue = *brightness value of the environment*
accelerationValue = *acceleration value of the user*
directionValue = *direction info of the user*

longtitudeValue = *longtitude value of the user location*
lattitudeValue = *lattitude value of the user location*

SensorController
+brightnessValue:double +accelerationValue:double +directionInfo:double +longtitudeValue:double +lattitudeValue:double
+setBrightnessValue(double br):void +getBrightnessValue() : double +setAccelerationValue(double br):void +getAccelerationValue() : double +setDirectionInfo(double br):void +getDirectionInfo() : double +setLongtitudeValue(double br):void +getLongtitudeValue() : double +setLattitudeValue(double br):void +getLattitudeValue() : double

5. SYSTEM ARCHITECTURE

5.1 Architectural Design



5.2. Description of Components

In this part, we don't mention the fourth part of each subsystem, because we think that they are complex for initial design report and we decide that these parts are included in detailed design report.

5.2.1 MainServerCore

5.2.1.1 Processing narrative for MainServerCore

Main Server Core is one of the major parts of our system. Actually it is responsible for connecting MDA with the database part. When MDA needs some info that is stored on database, MS Core analyzes this request and gets the data from database. Moreover, it has some manager units in order to handle Mission, Message and Map related issues. For example, when MDA needs info about a mission or message, MS Core analyzes the request, get the necessary data from database and return this info to MDA.

5.2.1.2 MainServerCore interface description

Input interfaces of this software component are explained in two parts. First part includes query results that are sent by database part. The second part includes user identification, message info and some data that should be written to database (For example; Mission or Unit object). These units are sent by MDA and processed by MS Core. The output interfaces are also considered as two main titles. First of all, MS Core sends the authentication report and filtered data to MDA. Secondly, it sends processed message or info and information query to database.

5.2.1.3 MainServerCore processing detail

In order to explain algorithmic description of this software unit, we should mention about the relation between MS Core, MDA and database. The processing detail can be explained as the following:

- MDA sends data to MS Core via the Mobile Transmission Unit for reading or writing it to database.
- MS Core analyzes this data and uses one of its manager parts in order to answer the request.
- Database processes the query that comes from MS Core and gives the necessary information to it.
- MS Core sends the filtered information back to the MDA by using again Mobile Transmission Unit.

5.2.2 DatabaseManager

5.2.2.1 Processing narrative for DatabaseManager

DatabaseManager is actually a part of MainServerCore unit. This unit handles database related jobs of MS Core part. At any time of our application, MDA may need some object data that is stored in database. At this point DatabaseManager part calls its necessary method in order to get filtered data from database. After that, this data is sent to MDA by MS Core.

5.2.2.2 DatabaseManager interface description

As explained the previous part, DatabaseManager gets data and queries this data in order to get the expected results. One of the input interfaces of this unit is id number of user. For example if the system needs the missions of the user, it gets id number and calls its getMission method. Then database returns Mission object that is the second input interface of DatabaseManager. Output interfaces are similar with input interfaces. First output interface is the id number of the user that is sent to database. Second one is the result of the database (for example; Mission object) that should be sent to MDA via MS Core.

5.2.2.3 DatabaseManager processing detail

The processing detail of DatabaseManager unit can be explained as the following:

- This unit gets id number of user that was sent by MDA.
- Then, it sends the data to database part by calling one of its methods.
- After querying the data, the results are sent to DatabaseManager.
- Finally, DatabaseManager returns back the filtered data to MDA part.

5.2.3 MissionManager

5.2.3.1 Processing narrative for MissionManager

MissionManager is actually a part of MainServerCore unit. This unit handles Mission related jobs of MS Core part. At any time of our application, MDA may need a Mission object of a user that is stored in database. At this point MissionManager part calls its necessary method in order to get filtered data from database. After that, this data is sent to MDA by MS Core.

5.2.3.2 MissionManager interface description

As explained the previous part, MissionManager gets data and queries this data in order to get the expected results. Moreover it also records a new Mission in database. One of the input interfaces of this unit is id number of user. For example if the system needs the missions of the user, it gets id number and calls its getMissionsOfUser method. Then database returns

Mission object that is the second input interface of DatabaseManager. The last input interface is the Mission object that comes from MobileApplicationCore. Output interfaces are similar with input interfaces. First output interface is the id number of the user that is sent to database. Second output interface is Mission object that is recorded to database. Finally, the last one is the result of the database (for example; Mission objects) that should be sent to MDA via MS Core.

5.2.3.3 MissionManager processing detail

The processing detail of DatabaseManager unit can be explained as the following:

- This unit gets id number of user that was sent by MDA.
- This unit gets a new Mission object that is recorded to database.
- It sends the id number to database part by calling one of its methods.
- It sends new Mission object to database by calling one of its methods.
- For user id case, after querying the data, the results are sent to MissionManager.
- For new Mission case, Mission object is recorded to database.
- Finally, DatabaseManager returns back the filtered data that is Mission objects that belong to the user id to MDA part.

5.2.4 MessageManager

5.2.4.1 Processing narrative for MessageManager

MessageManager is also a part of MainServerCore unit. This unit handles Message related jobs for MobileApplicationCore. At any time of our application, MDA may need messages of a particular user or all users. The system may need the unread messages of some users. Moreover we want to save messages to database in order keep track of them. For all of these issues, MessageManager part calls its necessary methods in order to get filtered data from database. After that, this data is sent to MDA by MS Core.

5.2.4.2 MessageManager interface description

Input interfaces of MessageManager can be categorized under three parts. First of all, the user id that comes from MobileApplicationCore. Second input interface is the Message objects that are returned from database. The final input interface is also Message object, but this object comes from MobileApplicationCore in order to save it to database. The output interfaces are categorized under two parts. One of them is user id that will be sent to database in order to query the data. Second output interface is Message objects that was given from database and that will be sent to MobileApplicationCore.

5.2.4.3 MessageManager processing detail

- Get user id for querying.
- Get Message object in order to save them into database.
- Either sends user id to database and get the expected results or sends Message object to the database in order to save it.
- If user id is sent, receive the required Message objects from database and then send these results to MobileApplicationCore.

5.2.5 MapManager

5.2.5.1 Processing narrative for MapManager

MapManager unit is responsible for displaying map on the screen. Since the maps are already stored our database and its size is much bigger than screen size, we decide to apply a logic that requires a sector implementation on map. In other words, we convert our map into many sectors and display it on the screen for specific sector number. This software unit has some methods in order to perform these desired actions.

5.2.5.2 MapManager interface description

One of the input interfaces of this software unit is coordinate of the user. The other input interface is Map object that comes from database according to a sector number. The output interfaces are analyzed in the same way but a little bit different. First output interface is sector number that is calculated according to the coordinate of the user. This number is sent to database in order to get required Map object. Second output interface is Map object that will be sent to MobileApplicationCore from MapManager.

5.2.5.3 MapManager processing detail

The processing detail of this unit is as the following:

- MapManager unit gets the location of the user from MobileApplicationCore. (MobileApplicationCore actually gets this info from SensorController)

- MapManager analyzes the coordinates of the user and defines the sector number that should be displayed on the screen.

- Since we know the specific sector number, MapManager returns Map object to

MobileApplicationCore according to this number.

5.2.6 AuthenticationManager

5.2.6.1 Processing narrative for AuthenticationManager

AuthenticationManager is actually a part of MS Core and responsible for user logins. It basically processes the user id and password and checks that if the password of the specific user is correct or not. Another job of this software unit is to record login history.

5.2.6.2 AuthenticationManager interface description

Input interfaces of this unit are user id and password of the user. Output interface is the boolean value that represents the user id and password are consistent and correct. These boolean value is sent to MobileApplicationCore.

5.2.6.3 AuthenticationManager processing detail

- Get user id and password from MobileApplicationCore.
- Check the correctness of these data by looking the values in the database.
- After checking database values, decide on the login operation.
- If the arguments are consistent return true and if the arguments are not consistent return false to the MobileApplicationCore.

5.2.7 TransmissionManager

5.2.7.1 Processing narrative for TransmissionManager

TransmissionManager unit is responsible to create communication between MS Core part and MobileApplicationCore part. When MS Core needs to send data to or receive data from MobileApplicationCore, this unit is activated and data transfer between these two units are handled.

5.2.7.2 TransmissionManager interface description

Input and output interfaces for this software unit are simpler than other units. When we think about input interfaces of this unit, we should think two separate parts. One of them is the data that we want to send to the MobileApplicationCore. For this aim, TransmissionManager gets the data from MS Core. The other input interface is data that comes from MobileApplicationCore in order to send data to MS Core. Output interfaces are just the

opposite way of input interfaces and have the same logic. Output interfaces are the data that is sent from TransmissionManager to MS Core and MobileApplicationCore.

5.2.7.3 TransmissionManager processing detail

Here is the processing detail of this unit:

- Get the data from MS Core.
- Hold this data in the unit for MobileApplicationCore.
- Sends the data to MobileApplicationCore.
- After MobileApplicationCore's operations, get the data from there in order to forward data to MS Core.
- Sends the required data to MS Core.

5.2.8 MobileApplicationCore

5.2.8.1 Processing narrative for MobileApplicationCore

Mobile application core is the unit that is in a relationship with User, MS Core and Sensor Controller. It interacts with user and enables him/her to use system applications. Moreover this unit is connected to Sensor Controller is order to answer the needs according to the CAUI. Finally, MobileApplicationCore communicates with MS Core and this enables to reach info of database and use it to handle some system functions.

5.2.8.2 MobileApplicationCore interface description

First of all, we talk about input interfaces of MobileApplicationCore. The first input interface is user touch screen inputs that are created by user. Second input interface is context reports that are sent from SensorController to this software unit. Finally the last input interface is the filtered data which is sent from MS Core unit. The output interfaces can be considered as the opposite ways of input interfaces and they are categorized in two parts. First output interface are objects that is sent to the User (actually device screen) in order to display the required data. Second and last output interface is data that is sent to MS Core in order to query it and get back the correct results.

5.2.8.3 MobileApplicationCore processing detail

Algorithmic description of this software unit can be analyzed as the following:

- It gets user inputs from touch screen.

- It analyzes the data and the request of the user.
- According to the request, it sends the necessary data to MS Core part
- After MS Core finishes its job, MobileApplicationCore gets the filtered data from it.
- It uses this filtered data and change user interface according to the request.

By the way, this unit gets the context report from SensorController and performs the necessary actions (changing user interface) according to this report continuously.

5.2.9 InputController

5.2.9.1 Processing narrative for InputController

InputController is actually a part of MobileApplicationCore and handles the input related issues. First of all it gets and understands the user inputs and analyzes them in order to find the requested actions. Moreover, this unit is in a relationship with UserInterfaceControler unit in order to adjust the interface settings.

5.2.9.2 InputController interface description

Input interface of this unit is user inputs that can be created by touch screen. InputController gets this input and analyzes them. After that the output interface of this unit plays an important role. Actually the output interface is the process info that is requested by the user. In other words InputContoller sends the user request to UserInterfaceController by analyzing the inputs.

5.2.9.3 InputController processing detail

Algorithmic description of this unit is as the following:

- Get the user inputs that is given by touch screen.
- Analyze the input and decide the process that is requested.
-After deciding this process, give this info to UserInterfaceController in order to display the requested actions on the screen.

5.2.10 MobileDeviceTransmissionUnit

5.2.10.1 Processing narrative for MobileDeviceTransmissionUnit

MobileDeviceTransmissionUnit is a unit that enables the connection between

MobileApplicationCore and MS Core. When mobile application needs to send data to MS or receive data from MS, this unit is activated and used to communicate these two separate parts.

5.2.10.2 MobileDeviceTransmissionUnit interface description

Input and output interfaces for this software unit are simpler than other units. When we think about input interfaces of this unit, we should think two separate things. One of them is the data that we want to send to the MS Core. For this aim, MobileDeviceTransmissionUnit gets the data from MobileApplicationCore. The other input interface is data that comes from MS Core in order to send data to MobileApplicationCore. Output interfaces are just the opposite way of input interfaces and have the same logic. Output interfaces are the data that is sent from MobileDeviceTransmissionUnit to MS Core and MobileApplicationCore.

5.2.10.3 MobileDeviceTransmissionUnit processing detail

Here is the processing detail of this unit:

- Get the data from MobileApplicationCore
- Hold this data in the unit for MS Core.
- Send the data to MS Core.
- After MS Core's operations, get the data from there in order to forward data to MobileApplicationCore.
- Send the required data to MobileApplicationCore.

5.2.11 UserInterfaceController

5.2.11.1 Processing narrative for UserInterfaceController

This unit controls the main operations of the UI and displays the required objects and texts to the screen. Many basic operations of the UI are handled by this unit, for example; displaying map, missions, messages etc. Moreover the icons on the screen are also created and displayed by this unit.

5.2.11.2 UserInterfaceController interface description

UserInterfaceController works as a bridge between User, UserInterfaceSettingsManager, InputController and Mobile Core Application. By looking this info, firstly we can analyze the input interfaces of this unit. First input interface is the data that comes from Mobile Core Application and InputController in order to display it on the screen. Second input interface is

the settings info that comes from UserInterfaceSettingsManager in order to resize the icons or manage the UI. However, Output interface of this unit is only related with the User. It creates and designs UI for the user and displays the necessary info on the screen.

5.2.11.3 UserInterfaceController processing detail

- It gets the user inputs from InputController and Mobile Core Application.
- It gets UI settings' info that comes from UserInterfaceSettingsManager.
- Analyze these data and display the necessary and required parts on the screen.

5.2.12 UserInterfaceSettingsManager

5.2.12.1 Processing narrative for UserInterfaceSettingsManager

The main responsibility of this unit is to adjust the size and view of the icons and map. For many reasons, the user may want to resize the map and icons or change the order of the icons. This unit has necessary methods to perform such tasks.

5.2.12.2 UserInterfaceSettingsManager interface description

The input interface of this unit can be considered as user inputs. According to these values, UI is reorganized. The output interface is again interacts with user. It is explained as new and changed UI that will be displayed according to the user needs.

5.2.12.3 UserInterfaceSettingsManager processing detail

Here is the processing detail of this unit:

- Get the required parameters from the user in order to reorganize UI.
- After getting these values create new forms of screen objects.
- After forming these new objects, display them on the screen and by this way, offer a new and desired UI to the user.

5.2.13 SensorController

5.2.13.1 Processing narrative for SensorController

SensorController unit is mainly responsible for collecting the environment variables and

reporting them to the MobileApplicationCore. It gets light, acceleration or direction info from some sensor and accelerometer and forwards these values to MobileApplicationCore. It is an important software unit in order to perform CAUI logic to our system.

5.2.13.2 SensorController interface description

Input interfaces of this unit come from environment, accelerometer and satellite. These interfaces can be explained as light, acceleration, direction and position info. The output interface of this unit is related with the Mobile Core Application. In other words, it sends the context report to MobileApplicationCore as this is the output interface of this unit.

5.2.13.3 SensorController processing detail

- Gets light information from light sensor.
- Gets acceleration and direction info from accelerometer
- Gets position info from satellite.
- After collecting the needed data sends them to MobileApplicationCore in order to perform the necessary actions.

5.3. Design Rationale

When we think our design properties and create the general architecture of our system, we consider some critical points that can affect our system for future developments. By looking architectural design of our project, it can be said that there are three main parts that compose our architectural design. These parts are namely, Mobile Application Core, Main Server Core and Database Manager. Actually, before we develop our design, we think to create a direct communication between mobile devices. However, we decide that integrating a main server to the system enables us to control it by managing main part and connecting all the mobile devices to one main server. Moreover we think that the total cost decreases, because connecting each mobile device can be long price. After that, we create a strong relation between Database Manager and Main Server Core parts. Since between these two parts, there will be big data flow during our implementations. Finally, in order to connect Mobile Application Core to the Main Server Core, we define a transmission system that connects these two parts. Following these steps, we are able to create the big picture and our architectural design.

6. User Interface Design

6.1. Overview of the User Interface

6.1.1. Design Considerations

While designing the user interface, we prioritize simplicity, flexibility and ease of use which is main aim of the project. The user interface is designed with flexible item that are capable of being located at different places, can be used in different sizes and colors. Whole purpose of the project is to maximize the information to be gathered from the screen in unit time. In other words providing a productive, effective and user friendly interface to find what you are looking for immediately. Therefore, we keep main theme (in this case the map) in the middle of the screen and place other objects around it. Moreover, we will enable user to do it any way he likes.

6.1.2. Functionality

The user interface is capable of giving commands to the program and receiving feedback with different functionalities.

6.1.2.1. Giving Input

User can interact with the interface with classic touch screen input methods such as tapping, dragging to use the buttons and icons and with the native keyboard of the device to enter characters to write messages or reports.

6.1.2.2. Browsing the Map

User can tap to zoom in and out and drag the map to navigate where he is looking for.

6.1.2.3. Selecting Items on the Map

Tapping on a unit or a place item will open the information box next to the icon.

6.1.2.4. Applying Preferences

By tapping onto the Settings button user can access to the settings window where he can change or set preferences about the user interface and the main program.

6.1.2.5. Changing Theme

The application is built to demonstrate the adaptive capabilities of the user interface according to changes. Therefore it is capable of changing colors, icons, buttons, fonts, object and font sizes whenever and wherever necessary.

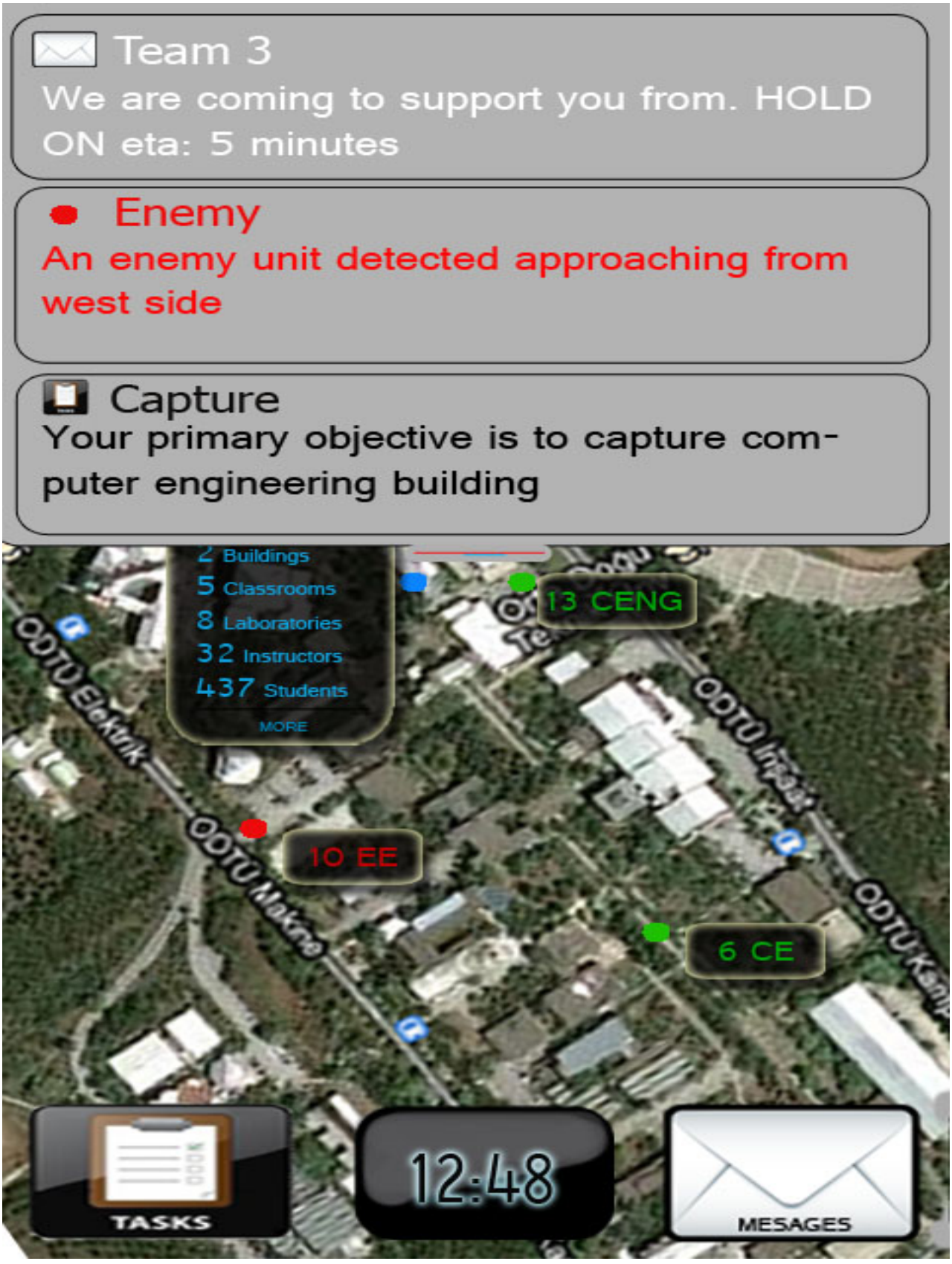
6.2. Screen Images

6.2.1. Main Window Image

6.2.1.1. While Standing

6.2.1.2. While Running

6.2.2. Displaying Notifications



6.2.3. Changing Layout

6.3. Screen Objects and Actions

6.3.1. Notifications Bar

Notifications bar will be displayed on top of the screen. Whenever a new task appointed, a new message arrives to user or an update concerning the user received an icon indicating the type and priority and the title of the received message will be displayed on the Notifications bar and will stay there until the user reads them. User can read the messages by tapping the area or dragging the bar to the bottom of the screen. This will open a new window with more details on messages and shortcuts to related parts of the application such as messages or missions.

6.3.2. Settings Button

Settings button opens a new window where user can alter the preferences or set new ones about the main application or the user interface. He can set what kind of messages he wants to see on the notifications area, how detailed he wants to see information, which theme he wants to use, what font size he prefers etc. by using classic check box and slider options

6.3.3. Layout Button

This button enables user to change the positions of the buttons. When user presses Layout button, all available buttons on the screen will display frames around them implying “you can move me”. User will tap and drag the button where he wants to and release. Then re-tapping Layout button will save the layout.

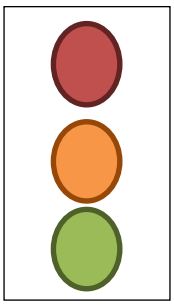
6.3.4 Missions (Tasks) Button

Tapping on Tasks button will open Missions Window where user will be able to see assigned missions to him. In this window user will have the options of reading missions, marking them complete or the opposite, sending reports about missions, sorting them according to priority or deadline etc.

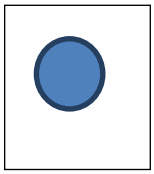
6.3.5. Messages Button

Tapping on Messages button will open the Messages Window. There, he can use basic messaging features such as reading or writing messages, tagging, forwarding, etc. Moreover, he can use archives of sent or received messages. While writing messages user will use the native keyboard of the device.

6.3.6. Unit Icons



The units will be shown with dots of different colors implying their alliances. We choose green for allies, yellow for neutrals and red for enemies. We will also display an alert mark for unknown threats or possible dangers meaning there may be an enemy unit at that location. Next to unit icons, some basic information will be displayed if available since intelligence reports on non-allies will be assumed limited. Tapping on the unit icon will display detailed knowledge and information about selected unit such as leader, members, technical abilities, military power and more.

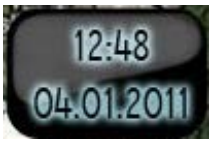


6.3.7. Place Icons

The map displays information about important locations on the map by default. The places are displayed by blue dots. In normal state, basic information about locations displayed next to this dot such as name of the place, population if it is a city or the length, altitude etc. if it is a passage. However, user can tap on the info area or the place icon (the dot) to display extra information about the place. This will include details such as hospitals, schools or military power of the city or intelligence reports about the passage, bridge etc.

6.3.8. Compass Icon

This icon shows the directions on the map. Therefore it helps user to understand which direction he is looking at or going to. In addition, this icon also works as a rotation button. The user may want to rotate the map or the displayed image on the screen. If so, he rotates the compass to rotate the map and compass still shows the right directions on the map.



6.3.9. Time

Time and date will also be displayed on the screen where user wants. According to the state of the user format of the date will vary from full “DD.MM.YY HH:MM” to “HH:MM:SS” or even a chronometer or a countdown clock. Tapping time object will open an agenda where user can see upcoming deadlines and tasks appointed to him.

6.3.10. Map

Map is the main object of the user interface where everything placed and designed according to. The idea is to extract data easily from the map therefore all the design considerations about user interface firstly based on an assumption that there is a map being displayed at background. It will cover the whole background. Font colors and sizes will be adapted also according to map and its main color histogram. User can be able to move the buttons to the empty parts of the map.

7. Detailed Design

7.1 MainServerCore

Classification
First of all, MainServerCore is a class. It includes the necessary manager objects that enable the communication between Database and MDA.

Definition
Main Server Core is one of the major parts of our system. Actually it is responsible for connecting MDA with the database part. When MDA needs data from database or sends data to database, this component handles this issue and supplies the required communication by using the manager objects. (MessageManager, MissionManager etc.)

Responsibilities
Firstly, this component is responsible to manage the manager objects. When the application runs, many functionalities of the application are used. In order to handle them, the manager objects takes and analyzes the data that comes from both MDA and database. At this point, MainServerCore is responsible for managing these objects and creating the continuous flow of data between MDA and database part.

Constraints
First of all, we talk about timing constraint of MainServerCore. As we explained before, it enables the communication between MDA and database part. However, when the user asks some data from database, he should get the required results in short time. Time of these operations should not take much time, because in this case the UI will not be useful and desirable. For example; when the user wants to see the messages that was received, MainServerCore should be fast enough and the results should be displayed on the screen at most a few seconds. This timing constraint is also valid for other operations such as; receiving the missions from database, sending the message info to database etc.

Composition

As we explained before, MainServerCore includes the manager components and controls them. Therefore, we can say that each of these manager components is subcomponent of MainServerCore. For example; MessageManager manages the messages by forwarding the necessary message data to either MDA or database part or MissionManager is responsible for getting the required mission info from database and sending to MDA part. These manager parts that will be explained below part and they work under the control of MainServerCore.

- DatabaseManager
- MissionManager
- MessageManager
- MapManager
- AuthenticationManager
- TransmissionManager

Uses/Interactions

MainServerCore uses the classes listed below and has an instance of each of them:

1. MapManager
2. MessageManager
3. MissionManager
4. DatabaseManager
5. AuthenticationManager
6. TransmissionManager

MainServerCore maintains interactions with the classes listed above.

Resources

When we think about resources that are managed and affected by this entity, the main resource that should be considered is database. In the previous parts, we said that MainServerCore includes DatabaseManager and it directly affects the states and situation of database. The operations that are processed by MainServerCore may add new data to database, may change the content of it or even may get data from it. The structure of database is continuously changed by this component.

Processing

Since the responsibilities of this component is significant, there are important duties necessary to fulfill its responsibilities. First of all, in order to make any operation that is made by the user meaningful, MainServerCore should complete some tasks by its manager components. Let’s analyze the relationship between these objects and MainServerCore and explain each of them in terms of its functions and characteristics.

In our application, for many operations we need to get data from database or send data to it. In such cases, MainServerCore uses DatabaseManager and the related methods of DatabaseManager are used. For example; when we need data from database, MainServerCore makes DatabaseManager active and starts to use its methods. Moreover, the related manager

parts are also become active for this operation. (If we make operation on messages, MessageManager is used and etc.) The possible scenarios are explained in the following part. (In the scenarios, only the parts that are related to MainServerCore are explained.)

Scenario 1: User wants to see the messages that were sent to him.
- Get the request from TransmissionManager.
- Analyze the request and call the necessary method in MessageManager. (For example; getReceivedMessages(int userId))
- DatabaseManager becomes active and makes the required queries.
- MessageManager obtains the results from database.
- TransmissionManager gets the results from MessageManager. (part of MainServerCore)

Scenario 2: Showing map on the screen
- TransmissionManager sends the request to the MainServerCore.
- Analyze the request and call the necessary method in MapManager. (For example; getMap(int sectorNumber))
- DatabaseManager gets the required map that is stored in database.
- MapManager gets the map from database.
- MainServerCore makes TransmissionManager active and it gets the results from MapManager.

* In this scenario, if the sectorNumber that is used by MapManager is not suitable, the exception is thrown that shows that the sector number is not correct.

Scenario 3: User wants to see the missions that were assigned to him.
- Get the request from TransmissionManager.
- Analyze the request and call the necessary method in MissionManager. (For example; getAssignedMissions(int userId))
- DatabaseManager becomes active and makes the required queries.
- MissionManager obtains the results from database.
- TransmissionManager gets the results from MissionManager. (part of MainServerCore)

Interface/Exports:

Database

This component interacts with database during the application.

Map

MainServerCore is connected to MapManager in our design and it provides Map objects for application.

Mission

MissionManager interacts with MainServerCore in our application in order to handle mission related objects. Therefore, this component provides Mission objects.

Message

MainServerCore and MessageManager are interacts and communicates with each other in order to manage the issues that are related with messages. In other words, MainServerCore provides Message objects.

7.2 DatabaseManager

Classification

DatabaseManager is a class.

Definition

DatabaseManager is a class that is responsible for handling database related issues. Getting necessary data from database, sending and storing new data to it or changing the content of it are some actions that are made under the control of DatabaseManager. In the application, there are a lot of operations that are related to database and that affect it. Therefore, we think that designing a manager that is responsible for database operations is important and necessary.

Responsibilities

Although there are many responsibilities of this component, we may categorize them under the following titles.

Getting Data from Database

Getting data from database is one of the major and important part of our application. Since many data are stored in database and they are needed during the application, we need to access the data in it at any time. At this point, the importance of DatabaseManager can be understood more easily. The methods and implementations that can accomplish this task lies in the structure of this component. It can query the database according to the request that comes from MDA and get the results from database. This query may be related with map, messages or missions.

Writing Data to Database

The logic of writing data to database is similar with the previous part that is getting data from database. This time data are written to database in order to satisfy user requests or application needs. For example; a mission that is assigned to a user may change and for this operation, DatabaseManager should be responsible for querying and writing the data to database. This writing operation is also valid for other operations like map or message operations.

Constraints

Timing is a critical constraint for DatabaseManager. Since the results that are obtained from database are directly used for restructuring UI and showed to the user, DatabaseManager should work fast and accurate. The slow operations of DatabaseManager means that the user gets the results and analyze them after long time. In other words, if the results of user requests

are displayed in UI after more than 1-2 seconds, it is not meaningful and suitable. Therefore this component should work under strict time constraints for the fluency of our application.

Composition

There is no subcomponent of DatabaseManager.

Uses/Interactions

This class has an instance used by MainServerCore in order to get values from database and write values to database.

Resources

As this component's name implies, the resource that is related with this component is database. It has direct access of it and can change the structure of it. In the previous sections, we explain that DatabaseManager can read from database or write to it. The structure of database is continuously changed by this component.

Processing

First of all DatabaseManager is mainly responsible for creating the connection with database. When the application stars, the connection with database should be created in order to complete the main tasks in application. Let's view and analyze the processing of this component.
- When the application starts, this component is responsible for creating connection with database.

During the application:

- The read or write request that is given from MDA comes to this component.
- DatabaseManager analyzes the request and it either queries to database and get the expected results or writes the obtained data to database.
- After the operation, this component forwards the result of database operation to MDA by using TransmissionManager of MainServerCore.

Interface/Exports

<u>Database</u>
This component interacts with database during the application

<u>Connection</u>

This component provides a Connection object in order to make connection with database.

DriverManager
DatabaseManager component provides this class in order to get connection.

SQLException
We use this exception class in order to handle DatabaseManager operations in correct and safety way.

Statement
This component uses this class' object in order to create statement on Connection object and make queries to database.

ResultSet
This class is used for this component in order to get query results from database.

7.3 MissionManager

Classification
MissionManager is one of the important classes for our system. In this class, data objects for information about missions can be sent or received between MDA with MS Core.

Definition
MissionManager is actually a part of MainServerCore that handles Mission related jobs of MS Core part. During processing of our application, MDA may need a Mission object of a user stored in database. At this point MissionManager part calls its necessary method in order to get filtered data from database. After that, this data is sent to MDA by MS Core.

Responsibilities
The main duty of this component is to manage Mission objects transfer between MDA with MS Core. MissionManager is responsible for getting data, querying this data and recording a new Mission in database. This class can provide the missions of any user if the system needs. Then, it can get Mission object that comes from MobileApplicationCore.

Constraints
Although we have not specific constraint for this component yet, we have some assumptions. For this component, we assume that when the user sends the request, our TransmissionManager and UserInterfaceManager works correctly and sends the request to this component.

Composition

Mission is base class. There are five different types defined missions called Arrest, Attack, Fire, Follow and Move classes as default. They are not formally categorized but they can be thought that like that:

Arrest mission can be thought a mission that a specific enemy should be arrested and the arrested time should be kept.

Fire and Attack missions are the missions that users need to destroy the enemy units. Fire mission can be thought that user perform it specified location with respect to coordinates of enemy.

Move mission can be thought that a mission is just for changing position from one coordinate to another.

Uses/Interactions

This class has an instance used by MainServerCore which used for handling the issues related to missions.

Resources:

The most important resource of the system is database. MissionManager class needs to use database as a resource. Missions sent to the users should be database. All missions of each user are recorded in database. Moreover, each mission has some attributes such as type of mission, rank of mission, privacy of mission and deadline of the mission. All these information is very important for the system and they should be in database and they should be accessible dynamically.

Processing

The other important part is how to MissionManager components go about performing the duties necessary to fulfill its responsibilities. Our scenario will be processed with some steps lie that:

- This class gets id number of user that was sent by MDA and gets a new Mission object that is recorded to database for the user with this id number.

- MissionManager sends the id number to database part by calling one of its methods and sends new Mission object to database by calling one of its methods.

- After querying the data, the result of the control is sent to the MissionManager.

- That new Mission is recorded to the database for specified user. The attributes of Mission object is processed. Name, rank, privacy and deadline are the attributes which is processed in Mission object for MissionManager.

- After all these steps, DatabaseManager sends back the filtered data which is Mission object that belongs to the user id to MDA part.

Interface/Exports

Mission

This component provides Mission class' objects for our application. These objects' methods that are used for handling mission related issues are explained as the following:

- recordMissiontoDB (Mission newMission) : void
- getMissionsOfUser (int userID) : string

Types

By looking the methods that are explained in the previous title, we can say that this component provides the following data types:

- Mission -> newMission //for defining new Mission object.
- int -> userID //for userID.
- String -> returnValue //for mission explanations.

Exceptions

Some methods of this component find and returns the missions that are belong to the user. These methods take this user's id as an argument and checks that whether the user id is valid or not. If there is no user that has this user id, this component throws an exception. For example;

-throw new MyException("illegal user id");

(after defining our Exception class, namely, "MyException").

7.4 MessageManager

Classification

MessageManager is a class that works under the control of MainServerCore.

Definition

MessageManager is one of the manager classes that works under the control of MainServerCore. As its name applies, this component handles message related operations for our application. First of all, a user can be send to messages to or receive messages from someone else. He should view the received and sent messages in our application by using UI. For these operations, the main component of our system is MessageManager.

Responsibilities

The responsibilities of this component can be analyzed as the following. First of all it works as a bridge that forwards messages between MDA and database part. All of the messages will be stored in database. Moreover when the user wants to see the received messages, DatabaseManager understands the request(with the assistance of TransmissionManager) and queries to the database in order to get the received messages. This logic is also valid for displaying sent messages. The mechanism is same with the previous one. Furthermore, for deleting messages of a user, DatabaseManager again becomes active and deletes the messages in the databases.

Constraints

Although we have not specific constraint for this component yet, we have some assumptions. We assume that when the user sends the request, our UserInterfaceManager and TransmissionManager works correctly and forward the request to this component.

Composition

There is no subcomponent of MessageManager.

Uses/Interactions

This class has an instance used by MainServerCore in order to accomplish the message related jobs of application.

Resources

The main resource that is affected by this component is database, because it affects the structure of database by adding or deleting the messages in it.

Processing

Now, we analyze the processing of this component:

- First of all, this component needs to understand the request that comes from user.
- After the user touches to the screen, UserInterfaceManager and TransmissionManager send the request to the MessageManger.
- MessageManager analyzes it and behaves according to the request.

- It contacts with DatabaseManager and make query according to the user needs.
- It makes the necessary operations on database and if any results are returned from database, it gets them.
- If any results need to be displayed to the user, it again contacts with TransmissionManager and forwards the result for the UI.

Interface/Exports

Message

This component provides Message class' objects for our application. These objects' methods that are used for handling message related issues that are explained as the following:

- recordMessagetoDB (Message newMessage) : void
- getMesaagestoUser (int UserID) : string
- getMessagesfromUser (int UserID) : string
- getMessagestoAll() : string
- getUnreadMessagestoAll () : string
- getUnreadMessagestoUser (int userID) : string

Types

By looking the methods that are explained in the previous title, we can say that this component provides the following data types:

- int -> UserID //for defining user id.
- Message -> newMessage //for defining the message object.
- String -> returnValue //for defining return values of methods.

Exceptions

Some methods of this component find and return the messages that are belong to the specific user. These methods take this user's id as an argument and checks that whether the user id is valid or not. If there is no user that has this user id, this component throws an exception. For example;

-throw new MyException("illegal user id");

(after defining our Exception class, namely, "MyException").

7.5 MapManager

Classification

MapManager is a class that is one of the manager classes that works under MainServerCore.

Definition

MapManager is one of the manager classes that works under the control of MainServerCore. It handles map related operations for our application. First of all, the maps that will be displayed on UI are stored in database. When, we need to show some part of the map on the

screen, we should somehow bring that part from database and display it to the user. For this purpose, MapManager is used for our application. It creates connection between MDA and database for map-related issues.

Responsibilities

There are some responsibilities of this component. First of all, it is responsible for converting map coordinates into the sectors. We should make this operation in order to bring the related part of the map on the screen. The other responsibility is getting the required part of the map according to the sector number that was received as an argument from MDA. These two main responsibilities supplies us the power in order to control the map operations in our application.

Constraints

We have an assumption for MapManager. We explained in the previous parts that we store the maps in database and MapManager helps us to display them on the screen. Therefore we assume that maps are in the database and we can reach them at any time during the application.

Composition

There is no subcomponent of MapManager.

Uses/Interactions

This class has an instance used by MainServerCore in order to display map on the screen.

Resources

The main resource that is used by this component is database. Although this component is not directly affects the structure of database, it uses the data and map information in order to display map on UI.

Processing

Now, we analyze the processing of this component:
- When the user starts our application, firstly MDA makes a request that it needs a map in order to display it on the screen. Then, this request comes to MapManger with the assistance of TransmissionManager.
- MapManager converts the coordinates into sectors for the main map that is stored in our

database.
- It gets the sub-part of the map from database and gives it to the TransmissionManager. (In order to display map on the UI)
- If the user wants to see the other parts of the map, the inputs of the screen is analyzed by InputController and comes to MapManager
-According to this new request MapManager repeats its actions that is needed to query the map into the database and to give it to the MDA part.

Interface/Exports

Map
This component provides Map class' objects for our application. They have methods which are used for handling map related issues that are explained as the following:
- convertCoordinatestoSector(float altitude, float latitude) : int
- getMap (int sectorNumber) : Map

Types
By looking the methods that are explained in the previous title, we can say that this component provides the following data types:
- float -> altitude, latitude //for representing coordinate values.
- int -> sectorNumber //for holding sector number
- Map ->returnValue //for returning Map object

Exceptions
This component provides and uses exceptions. If a map request that comes from MDA is not valid, MissionManager object throws an exception that shows that the request is not valid. For example after defining our Exception class, namely, "MyException":
-throw new MyException("illegal map request");

7.6 MobileApplicationCore

Classification

MobileDeviceCore is a class that is the center the mobile device application.

Definition

This class is responsible for managing mobile device application with controlling instances of appropriate classes.

Responsibilities

The first responsibility is related with managing the user inputs with the assistance of

UserInterfaceManager. This component should understand and analyze the user inputs in order answer his requests. Second responsibility is getting the environment context from SensorManager and forwards it to the UserInterfaceManager. The other responsibility of this component is to create UI. For this purpose MobileApplicationCore and UserInterfaceManager work with together. As the last responsibility, this component should send the any data to TransmissionManager in order to interact with MainServerCore and database part.

Constraints

Firstly, we should talk about timing constraint of MobileApplicationCore. This component enables the communication between MDA and the user. However, it should response the user requests in a very short time. For example; when the user wants to see the messages that was received, the communication between MDA and UserInterfaceManager should be fast enough and the new UI should displayed on the screen at most a few seconds. This timing constraint is also valid for other operations such as; receiving the missions from database, changing the structure of UI according to the context info etc.

Composition

The followings are the classes that have instances as a member of the MobileDeviceCore class.

1. MobileApplicationCore
2. InputController
3. MobileDeviceTransmissionManager
4. UserInterfaceController
5. SettingsManager
6. SensorController
7. Mission
8. Message

MobileDeviceCore uses the methods of these classes to maintain communications between them and to manage the main application.

Uses/Interactions

MobileDeviceCore uses the classes listed below and has an instance of each of them:

1. InputController
2. MobileDeviceTransmissionManager
3. UserInterfaceController
4. SettingsManager
5. SensorController

6. Mission
7. Message

MobileDeviceCore maintains interactions with the classes listed above.

Resources

There are some resources that are needed by this component. First of all it needs the sensors that collects the context info and forwards them to MobileApplicationCore. This component needs this info, because the UI can be restructured by using them. Morover, the user input can be analyzed as another resource that are managed by this component. User inputs are very important metric for our application, because it behaves and runs according to the user requests.

Processing

Below are the possible interactions of the members of the MobileDeviceCore:
* SensorController - UserInterfaceController
* SettingsManager - UserInterfaceController
*UserInterfaceController - MobileDeviceTransmissionManager
Each has explained in specific sections of the corresponding classes.

Interface/Exports

This class has no interface of its own. It is only a container that connects the parts of the mobile application; therefore it uses interfaces of each subclass to connect one to another.

7.7 UserInterfaceController

Classification

UserInterfaceController is a class works under MobileApplicationCore.

Definition

MobileApplicationCore has an instance of this class as its member. This class specifies what to be displayed on screen and where and how they are displayed. Moreover this class handles user inputs and directs them to related classes. This is where the application interacts with the user.

Responsibilities

UserInterfaceController is responsible for generating the layout of the user interface and handling the user actions on screen. UserInterfaceController reads the context information data and user preferences from SensorController and SettingsManager respectively through the MobileApplicationcore. Then it combines this information to generate the user interface to be displayed on screen. Secondly, this class is responsible for delivering user input to its corresponding class. For example, when user writes a message through the message screen it is UserInterfaceController's job to set the corresponding fields of the Message class or to call the related methods.

Constraints

This class has no constraints.

Composition

This class has no subcomponents. It interacts with other classes via MobileDeviceCore.

Uses/Interactions

This class is used by MobileDeviceCore to generate and display the user interface. It gets data from SensorController and SettingsManager instances and sends data to Mission and Message instances of the MobileDeviceCore.

Resources

They are no memory or data resources directly affected by this class. It only changes the existing variables and sets different values. It uses central processor and graphical processing units inside the mobile device.

Processing

We can divide this into two subsections: firstly, generating the user interface; secondly, delivering user input. The primary object of the project is to provide a user-friendly graphical interface. Therefore, user choices and sensor outputs are taken into consideration is building the interface. UserInterfaceController gathers data from both sources then combines them to generate the desired interface with desired fonts, buttons, sizes, colors and layout. Secondly, user interacts with the application via graphical interface. Therefore, when he wants to enter a text or write a message, UserInterfaceController calls the corresponding method. For example, when he writes a message, UserInterfaceControoler calls setMessageSubject(text), setReceivers(user names), setMessageBody(message) etc.

7.8 SettingsManager

Classification

SettingsManager is a class that has an instance constructed by MobileDeviceCore.

Definition

This class manages changes desired by user about both the core application and the graphical user interface.

Responsibilities

This class is responsible for managing the settings and storing preferences. They may be either about the user interface or about the application behind the interface. This class provides access to its components via getters for each of them.

Constraints

User can reach SettingsManager when he is not in an emergency situation in which the application will block the settings menu.

Composition

There is no subcomponent of SettingsManager.

Uses/Interactions

SettingsManager has an instance under the MobileApplicationCore. This instance kepps the preferences set by user and provides them as asked by MobileApplicaitonCore or the UserInterfaceController by getter methods defined in itself.

Resources

There are no resources directly affected by this class since it only sets values of variable already in the memory.

Processing

SettingsManager becomes active when user opens settings menu by tapping the settings icon on the screen. The menu pops up with two tabs oneis for user interface and the other is for genereal settings. User can either change the menu by selecting a different tab or sets a

preference value by selecting preference item. Later these settings will be read by UserInterfaceController to display the interface as desired or by the MobileApplicationCore to set the actions of application as desired.

Interface/Exports

This class contains two sets of preferences. Each will be displayed as a different tab in the GUI. These items have their fields as members of the class. Below is the list of these items:
User interface preferences:

-fontType
-fontSize
-fontColor
-buttonSize
-backgroundColor
-informationBoxColor
-informationBoxOpacity
-displayAllies
-displayNeutrals
-displayEnemies
-displayPlaces

Application preferences:
-time
-chronometer
-privacy
-notificationRank
-messageRank

7.9 SensorController

Classification

SensorManager is a class responsible for gathering sensor data to read the context and delivering them to the MobileDeviceCore.

Definition

SensorController class is the part of the project that generates the “context awarenes”. In other words, this is the part where device interacts with the environment and determine the situation and changes around itself.

Responsibilities

SensorController is responsible for interacting with four sensors and MobileDeviceCore. Main responsibilities of this class are to read sensor outputs, determine the context then delivering that information to MobileDeviceCore. In other words its job is to maintain communications between these parts.

Constraints

The only constraint of this class is to evaluate sensor outputs efficiently and deliver them to MobileDeviceCore in a short time to provide a fluent and friendly user interface.

Composition

There is no subcomponent of SensorController.

Uses/Interactions

This class has an instance used by MobileDeviceCore to get the context information read from the sensors.

Resources

The input data to this class are sensor output maintained by hardware. Since there is an abstraction, we do not need to deal with the details of sensors. Android platform provides necessary methods to interact with sensors.

Processing

This class is constructed by MobileDeviceCore. MobileDeviceCore calls the "getContextInfo()" method based on a timer (such as in each fifteen second give me your report). Then SensorController calls the sensor reader methods to get the data from the sensors and evaluate them. Evaluation will be based on result of the test applications for matching data with several conditions. We need the acceleration values that correspond to different movement styles or lightness values for different illumination conditions. Finally it returns the context information to the MobileDeviceCore.
This class has no need for a user interface. It works as a background process. The interface with SensorController and MobileDeviceCore is consists of just "getContextInfo()" method which is also the only path between them. The interface between this class and sensors is maintained by Android's native libraries.

Interface/Exports

Types:

-float -> illuminaitonValue
-float -> accelerationValue
-float -> coordinates
-float -> direction
-Context -> contextInfo

This class consists of reader methods for four sensors namely:

"readLightness()",
"readAcceleration()",
"readCoordinates()",
"readDirection()".

These methods set the corresponding fields of the class which are lightnessValue, accelerationValue, coordinates and direction respectively.
Moreover, it has a "contextEvaluation()" method that uses the four fields mentioned above to conclude a context information in which the device is working.
Finally, it also has a "getContextInfo()" method that is to be called by MobileDeviceCore to get the context information data.

8. LIBRARIES AND TOOLS

Making a choice for suitable tools and libraries to be able to develop our project efficiently is a very difficult process. For this reason, a wide research was made on the Internet to find suitable tools and libraries that can be used in the project.

8.1 Eclipse

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. Moreover, it is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins.
For Java developers, Eclipse consists of the Java Development Tools (JDT) and users can extend its abilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. The aspects of Eclipse have an important role about why we choose it to for development environment. We plan to use Android platform due to its usability for mobile devices with Eclipse.

8.2 Android

Android is the most suitable platform that can respond most of the necessities of the project. Therefore, we chose Android because it is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

8.2.1 android.hardware.SensorManager

It will be necessary to use sensors such as light sensor, accelerometer sensor etc. for this project. Android provides developers this library to implement these sensors in applications.

8.2.2 SQLite

In this project, database need to be used to store necessary data. Android also can provide a powerful and lightweight relational database engine available to all applications called SQLite.

8.2.3 Android Emulator

The emulator available in the Android SDK is not just a tool that allows you to easily test applications without having to install it to a real device, or even having one. With the proper configuration it is possible to test situations which are hardly reproduced on a physical one. Android emulator is almost the best tool existing to develop our mobile device project.

9. TIME PLANNING (GANNT CHART)

9.1 Term 1 Gannt Chart

Components/Task	Dependent Components	Status	Date Start	Data Complete	Owner	Difficulty	Notes
Context Aware User Interface Term 1							
Pre - Proposal Report	Proposal Report	done	12-Eki-2010	18/10/2010	Burak	Small	Use the status to toggle between different states with color coding.
Market Research	Proposal Report	done	19/10/2010	3.11.2010	Hüseyin	Medium	marketing research for our project topic and similar products to our product
Literature Survey	Proposal Report	done	20/10/2010	4.11.2010	Ender	Medium	researching example papers and articles for context awareness
Proposal Report	Requirement Analysis	done	21/10/2010	5.11.2010	All of team	Medium	proposal report of our project
Requirement Analysis	SRS	done	6.11.2010	4.12.2010	All of team	Large	Requirement analysis before preparing software requirement specification
Software Requirement Specification 5-Dec-2010							
Initial Research and installation of Android	Detailed Design	done	6-Ara-2010	10.12.2010	Burak	Small	Installation of Android and its plug-ins. Research of our project and how to implement our project
Initial Network Design	Detailed Design	done	6-Ara-2010	12.12.2010	Hüseyin	Small	how to be our design model in the project
Research for Server - Client Connection	Detailed Design	done	6-Ara-2010	12.12.2010	Ender	Medium	Server-client connection architecture for data connection
Initial User Interface	Detailed Design	done	10.12.2010	16/12/2010	Hüseyin, Ender	Medium	First user interface reflecting our interface opinion
Initial Database Infrastructure	Detailed Design	done	10.12.2010	16/12/2010	Burak	Medium	how to design and implement our database
Initial Design Report	DDR	done	6-Ara-2010	16/12/2010	All of team	Medium	First report for our detailed design report
Testing Software Design	DDR	done	6-Ara-2010	16/12/2010	All of team	Medium	we will test our design
Team Presentation	Prototype Demo	open	21/12/2010	3.1.2011	All of team	Medium	Team presentation will be hold in a specific date
Detailed Design	DDR	open	17/12/2010	4.1.2011	All ot team	Large	Before preparing our detailed design report, we will implement final detailed design
Detailed Design Report 4-Jan-2011							
First Prototype	Prototype Demo	future	5-Oca-2011	17/1/2011	All of team	Medium	we will develop our first prototype by using our detailed design report
Prototype Demo	Final Presentation	future	66 1/1/2011	23/1/2011	All of team	Large	Final presentation of our demo will be hold in a specific date
Final Presentation 20-Jan-2011							
Prototype Demo	Final Presentation	future	10/1/2011	23/1/2011	All of team	Large	Final presentation of our demo will be hold.

9.2 Term 2 Gannt Chart

Components/Task	Dependent Components	Status	Date Start	Data Complete	Owner	Difficulty	Notes
Context Aware User Interface Term 2							
Web Page Design for Our Project Page	Blog Management	open	25/12/2010	31-5-2011	Ender	Medium	
Blog Management		open	1-Şub-2011	31-5-2011	All	Medium	
Development and Tool Analysis		open	15-12-2009	16-2-2011	All	Medium	
Development and Tool Integration	**20-2-2010**						
Sensor Implementation to system		future	25-12/2010	30-2-2011	Burak	Small	
Semantic Zooming		future	20-1-2011	15-3-2011	All	Large	
Database architecture	Database Design Implementation	future	15-2-2011	15-3-2011	Hüseyin, Ender	Large	
Complete Menu		future	15-2-2011	15-3-2011	Burak	Medium	
GUI		future	25-2-2011	20-3-2011	All	Large	
Network Design		future	1-Mar-2011	10.4.2011	All	Large	
Scenario		future	15-3-2011	15-4-2011	Arjun	Small	
Database Implementation	**15-4-2010**						
Integration	Complete Product	future	15-4-2011	30-4-2011	All	Medium	
Testing	Complete Product	future	15-4-2011	25-4-2011	Hüseyin, Burak	Medium	
Optimization	Complete Product	future	25-4-2011	30-4-2011	Ender	Medium	
Test and Debugging	Complete Product	future	1-May-2011	10.5.2011	Hüseyin, Burak	Medium	
Complete Product		future	1-May-2011	30-5-2011	All	Large	
Final Demo		future	1-Haz-2011	15-6-2011	All	Large	
Finalization	**20-6-2010**						

10. CONCLUSION

This document expresses the design approach taken by Momo Software for CAUI project. In this document a fair amount of improvement has been done on both project scenario and design related issues. As we develop our design document the goals and boundaries of our final product has become more clear and well-defined. Moreover, this document is the first point that we develop and analyze our design. We discussed the general architecture of the system and gave further information on technical design. Finally we explained tools, libraries and future plans of the project. Therefore this document will be definitely used for developing and creating our product in future.

Although this document has a lot of important data that is required for our development phase, we have to analyze the existing concepts and think our implementation techniques in more detail in order to start the implementation of the product. Still, this document will help us in order to gain an understanding of what we code and what we might face during the implementation phase. In summary, it can be considered as a step that helps us to reach our goals and final product.