# TABLE OF CONTENTS

1. INTRODUCTION.................................................................................................................... 4  
   1.1. Purpose of This Document ............................................................................................ 4  
   1.2. Project Description ....................................................................................................... 4  
   1.3. Project Features ............................................................................................................ 4  
   1.4. Goals and Objectives ................................................................................................... 5  
   1.5. Modules of CoreAccess ................................................................................................ 5  
   1.6. Process Model ............................................................................................................... 6  
   1.7. Hardware and Software Requirements ....................................................................... 6  
      1.7.1 Hardware Requirements ......................................................................................... 6  
      1.7.2 Software Requirements ......................................................................................... 6  
2. DESIGN CONSTRAINTS........................................................................................................ 7  
   2.1. Time Constraints .......................................................................................................... 7  
   2.2. Hardware Constraints ................................................................................................ 7  
   2.3. Software Constraints .................................................................................................. 7  
   2.4. Performance Constraints ............................................................................................ 7  
3. MODELLING ....................................................................................................................... 8  
   3.1. Data Model .................................................................................................................. 8  
      3.1.1. Entity - Relationship Diagram .............................................................................. 8  
      3.1.2. Description of ER Tables .................................................................................. 10  
   3.2. Functional Model ......................................................................................................... 13  
      3.2.1. Data Flow Diagram ............................................................................................ 13  
         3.2.1.1. DFD Level 0 ................................................................................................. 13  
         3.2.1.2. DFD Level 1 ............................................................................................... 14  
         3.2.1.3. DFD Level 2 (1.0) ..................................................................................... 15  
         3.2.1.4. DFD Level 2 (2.0) ..................................................................................... 16  
         3.2.1.5. DFD Level 2 (3.0) ..................................................................................... 17  
         3.2.1.6. DFD Level 2 (4.0) ..................................................................................... 18  
         3.2.1.7. DFD Level 2 (5.0) ..................................................................................... 18  
         3.2.1.8. DFD Level 2 (6.0) ..................................................................................... 19  
         3.2.1.9. DFD Level 2 (7.0) ..................................................................................... 20  
   3.2.2. Data Dictionaries ..................................................................................................... 21  
4. OVERALL ARCHITECTURE ............................................................................................... 34  
   4.1. List of modules .............................................................................................................. 34  
   4.2. Architecture Diagram .................................................................................................. 35  
   4.3. Details of Modules ....................................................................................................... 36  
5. COREACCESS USER INTERFACE .................................................................................. 39  
   5.1. User Functionality ...................................................................................................... 39  
   5.2. Content Manager Functionality .................................................................................. 48  
   5.3. System Administrator Functionality ........................................................................... 50  
6. UML DIAGRAMS ............................................................................................................... 55  
   6.1. Use Case Diagrams ..................................................................................................... 55  
      6.1.1. Use Case Diagrams of the User .......................................................................... 55  
         6.1.1.1. Positioning Use Case .................................................................................... 55  
         6.1.1.2. Search by Attributes Use Case .................................................................... 55  
         6.1.1.3. Search by Category Use Case ..................................................................... 56  
         6.1.1.4. Search on Map Use Case .......................................................................... 57  
         6.1.1.5. Display Results Use Case .......................................................................... 58  
      6.1.2. Use Case Diagrams of the Administrator ............................................................. 59
6.1.2.1. Login Use Case ................................................................. 59
6.1.2.2. Manage Map Use Case ...................................................... 59
6.1.2.3. Manage Activity Place Use Case ....................................... 60
6.1.2.4. Manage Vehicle Use Case ................................................ 61
6.1.2.5. Manage Ontology Use Case ............................................. 61
6.1.2.6. Manage Content Managers Use Case .............................. 62
6.1.2.7. Log Use Case ................................................................. 62
6.1.3. Use Case Diagrams of the Content Manager ....................... 63
6.1.3.1. Login Use Case ............................................................... 63
6.1.3.2. Manage Activity Use Case ............................................... 64
6.1.3.3. Manage Activity Place Use Case ..................................... 64
6.2. Class Diagrams .................................................................... 66
6.2.1. PDA Class Diagram ........................................................... 66
6.2.2. Request Handler Class Diagram ........................................ 68
6.2.3. Activity Search Class Diagram .......................................... 70
6.2.4. Transportation Class Diagram .......................................... 71
6.2.5. Web Service Class Diagram ............................................. 72
6.2.6. Database Class Diagram .................................................. 72
6.3. Sequence Diagrams ............................................................. 73
6.3.1. Search by Attributes ......................................................... 73
6.3.2. Search by Category ........................................................... 75
6.3.3. Search on Map ................................................................. 76
6.3.4. Retrieve Position ............................................................. 77
  6.3.4.1. Via GPS Receiver .......................................................... 77
  6.3.4.2. Via Manually Entering ................................................... 78
  6.3.4.3. Via Browsing Map .......................................................... 78
6.3.5. See Detailed Information .................................................. 79
6.3.6. See Transportation on Map .............................................. 80
6.3.7. Web Service Communication .......................................... 81
  6.3.7.1. PDA Web Service .......................................................... 81
  6.3.7.2. Administrator Web Service ........................................... 81
  6.3.7.3. Content Manager Web Service ..................................... 82
6.3.8. Handle XML ................................................................. 82
6.3.9. Find Vehicles ................................................................. 83
6.3.10. Find Results ................................................................. 84
6.3.11. Content Manager Database ............................................ 85
6.3.12. Administrator Database ................................................. 85
6.4. Activity Diagrams ............................................................... 87
  6.4.1. User Activity Diagram ...................................................... 87
  6.4.2. Web Service Activity Diagram ......................................... 88
  6.4.3. Administrator and Content Manager Activity Diagram .......... 89
6.5. Collaboration Diagrams ...................................................... 90
  6.5.1. Search by Attributes ......................................................... 91
  6.5.2. Search by Category .......................................................... 92
  6.5.3. Search on Map ............................................................... 93
  6.5.4. Transportation Manipulation on Server Part ..................... 94
  6.5.5. Activity and Place on Server Part .................................... 95
  6.5.6. Administrator and Content Manager Interface Communication .......................................................... 96
7. SYNTAX DEFINITION ......................................................... 96
  7.1. XML File Representation .................................................. 96
7.2. XSD of the System ........................................................................................................ 101
8. CONCLUSION .................................................................................................................... 111
9. APPENDIX ....................................................................................................................... 111
  9.1. Updated Gantt Chart ................................................................................................... 111
1. INTRODUCTION

1.1. Purpose of This Document

After releasing the initial report for the product CoreAccess, CoreTech has spent no time to form a complete and detailed design report. By gaining adequate experience and gathering relevant feedback from our previous requirement analysis report and initial design report, we were ready for detailed design report. The main purpose of the detailed design report was to show all the design processes about the project before the implementation for the project to be efficient and stable. Like initial design, we included several diagrams to make the project clear and easy to understand.

CoreTech is aware that a successful project can only survive if it has a proper design. Keeping this principle in mind, we did hard work to make every point clear. By forming a high quality design report, we think we will not encounter any difficulties at the implementation phase at second semester.

1.2. Project Description

CoreAccess is a Mobile GIS (Geographic Information System) Application, mainly dealing with social activities. These social activity places include cinema, concert hall, theatre, sport centers and the transportation opportunities among these activities. After learning the position of the user (via GPS (Global Positioning System) receiver, via marking on map or just entering his / her position manually) and the position of the cultural activity he/she chose by using PDA, this information is collected and processed. Finally, the result is shown to the user visually with transportation alternatives.

1.3. Project Features

Our product CoreAccess will mainly provide the followings:

- **Search Activity or Place:** The user can search any specific category (cinema, theatre, concert hall, sport centers), any specific activity attribute (name, place, date, etc.) or just browse the map for any activity or place. In addition to these search facilities, user can make search by entering an address keyword for an address search.

- **Ontology Help:** One useful property which CoreAccess serves is the ontology. By the help of ontology, user will be offered some alternative activities, if he/she does an unsuccessful search. Surely a user, who faces with a set of alternatives related to his/her search item, will be happier than a user who just sees a message like “The item you searched is not found!!!”

- **Visual Result:** When a user does any search using CoreAccess, he/she will be faced with a visual result. User will be able to show his/her search items on the map, with a shortest path drown. Certainly, some useful operations such as rotating, scaling, zooming in/out, scrolling the map will be provided.

- **Transportation:** The results will come with several transportation alternatives. Between two points one vehicle is not the only solution. Combination of vehicles for the specific distance is also supplied. User will be given a list of transportation
combination alternatives. According to the selection of the user, details of the vehicles and their paths will be shown. Moreover, CoreAccess will make estimations about the time and cost among these alternatives. Without a doubt, this is a very advantageous property.

- **AI/Algorithms**: In transportation functionality, there will be severe use of shortest path and graph algorithms. A map consists of thousands of nodes or sometimes more, so our algorithms have to be efficient.

- **Web Services**: Platform independency of CoreAccess lies on Web Services mostly. Once we are successful in implementing our project with Web Services, our main application can be called from anywhere.

- **Multilingual Support**: CoreAccess will come with 5 languages, namely Turkish, English, German, French, and Spanish.

### 1.4. Goals and Objectives

While we are designing our project, these main goals and objectives are considered:

- Organizing a complete project with every aspects,
- Programming PDA effectively and user friendly,
- Making the PDA – Server connection via Web – Service,
- Implementing ontology for cinema and transportation,
- Generating and implementing effective shortest path algorithms,
- Manipulating map data in GIS part.

### 1.5. Modules of CoreAccess

We have decomposed CoreAccess into modules. These are:

- Web Service Module
- GUI Module
- Administrator Module
- Content Manager Module

We have constructed our UML diagrams around the concept of these modules. While expanding our design, data flow diagrams, use case diagrams and state transition diagrams of our analysis report helped us a lot. Written details of these modules will be given in detailed design report.
1.6. **Process Model**

We have chosen Iterative Model for CoreAccess as we mentioned at our analysis report. We thought we can release some prototypes at implementation stage and according to success of these prototypes we can return back to the design. Actually, we realized we have made the right decision to have chosen this model. We tried to do our design considering these matters.

1.7. **Hardware and Software Requirements**

1.7.1 **Hardware Requirements**

There is no change in our hardware requirements when compared with our analysis report. In this design phase, we have discovered the importance of amount RAM in our server machine because we will store the vector map in RAM for fast processing. We have proposed 1 GB DDRAM in the analysis report and this amount will be sufficient.

1.7.2 **Software Requirements**

In the initial design report, we have talked about an external product, namely GeoMedia. At that time, there were license problems but now we have the product. We have spent some days for discovering it. InterGraph employees had told us that API comes with Geomedia, however, the other MOBGIS groups and we have seen that GeoMedia is only a desktop application for drawing and managing vector maps. GeoMedia does not supply an API for usage. It stores maps in MS Access tables in a closed format. We will find out the structure of these tables and write our own map editor.

For the server side development, we are using .Net 2005, and our development language is C#. Development on the PDA side has no choice other than .Net for compatibility with Windows CE. We have developed a prototype of the prototype for our TA Oral Dalay on 5.1.2006. We have successfully developed a PDA application which uses the TerraServer (http://terraserver-usa.com) open web services which is a service of MSN. .Net Smart Device application meets our necessities very well. Our premature application can search a place and display the result in 24 different zoom levels for 4 different map types.
2. DESIGN CONSTRAINTS

2.1. Time Constraints

CoreAccess project has to be finished at the end of May 2006. Moreover, we have to release a prototype at the end of this semester. As long as, the group members follow the schedule, the project could be completed successfully.

2.2. Hardware Constraints

In our department, there are only 4 PDA’s for testing our application. However, we will use emulator for PDA programming part. It does not affect our implementation very much. Moreover, we have some concerns about the GPS data. Since we do not have a GPS device, we have to simulate some sort of GPS data for determining the position of the user. On the other hand, interpreting the GPS strings is a very trivial job. After the comments made about analysis report by the instructors, we have prepared a detailed GPS Research Report which can be accessed via our web page.

2.3. Software Constraints

We are planning to use GeoMedia which is developed by InterGraph Company. We had a meeting in which the product and its libraries for GIS data manipulation were introduced. At last, we have taken the license of GeoMedia. However, it does not have an api for PDA side. Therefore we are going to write the map manipulation functions and application for PDA user. If we could not find an appropriate vector map, we will design a map in GeoMedia by ourselves.

2.4. Performance Constraints

Performance constraints have two aspects, first the PDA side, second the server side. It is clear that PDA has limited capabilities. Because of this reason, we don’t let PDA do complex and tiring calculations, it only takes user requests and displays the responses to those requests. For the visual capabilities, since vector maps are light visual objects, we don’t expect any problems. On the server side, there will be complex calculations and lots of database interactions. Especially while finding transportation alternatives and shortest paths, there may be cases such that every node of a city has to be traversed. Due to this obstacle, when our server application is first initialized, city map will be loaded to RAM and never get out until map data is changed (behaves just like RAM DISK systems). By this way, disk access will be eliminated, which is the main source of slow calculations.
3. MODELLING

3.1. Data Model

3.1.1. Entity - Relationship Diagram
3.1.2. Description of ER Tables

Staff

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the staff</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>The name of the staff</td>
</tr>
<tr>
<td>Login Name</td>
<td>String</td>
<td>The name for login process</td>
</tr>
<tr>
<td>Password</td>
<td>String</td>
<td>The password for entering the system</td>
</tr>
<tr>
<td>Company</td>
<td>String</td>
<td>The company of the staff if he/she is a content manager</td>
</tr>
<tr>
<td>Salary</td>
<td>String</td>
<td>Salary of the staff</td>
</tr>
<tr>
<td>Registered Day</td>
<td>Date</td>
<td>The date of the register of the staff</td>
</tr>
</tbody>
</table>

Activity

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the activity</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>The name of the activity</td>
</tr>
<tr>
<td>Place</td>
<td>String</td>
<td>The place of the activity</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
<td>The date of the activity</td>
</tr>
<tr>
<td>Category</td>
<td>String</td>
<td>The category of the activity (cinema, sports)</td>
</tr>
<tr>
<td>Time</td>
<td>Time</td>
<td>The time of the activity</td>
</tr>
</tbody>
</table>

Activity Places

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the activity place</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>The name of the activity place</td>
</tr>
<tr>
<td>Address</td>
<td>String</td>
<td>The address of the activity place</td>
</tr>
<tr>
<td>Phone</td>
<td>String</td>
<td>The phone of the activity place</td>
</tr>
<tr>
<td>Latitude</td>
<td>Double</td>
<td>The latitude of the activity place</td>
</tr>
<tr>
<td>Longitude</td>
<td>Double</td>
<td>The longitude of the activity place</td>
</tr>
</tbody>
</table>

Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the type</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>The name of the activity type</td>
</tr>
</tbody>
</table>

Map

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the map</td>
</tr>
<tr>
<td>Borders_Up_X</td>
<td>Double</td>
<td>The up – x point of the border</td>
</tr>
<tr>
<td>Borders_Up_Y</td>
<td>Double</td>
<td>The up – y point of the border</td>
</tr>
<tr>
<td>Borders_Down_X</td>
<td>Double</td>
<td>The down – x point of the border</td>
</tr>
<tr>
<td>Borders_Down_Y</td>
<td>Double</td>
<td>The down – y point of the border</td>
</tr>
<tr>
<td>File_Path</td>
<td>String</td>
<td>The path information on the map</td>
</tr>
</tbody>
</table>
### Node

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the node</td>
</tr>
<tr>
<td>Latitude</td>
<td>Double</td>
<td>The latitude of the node</td>
</tr>
<tr>
<td>Longitude</td>
<td>Double</td>
<td>The longitude of the node</td>
</tr>
<tr>
<td>Type</td>
<td>Double</td>
<td>The type of the node</td>
</tr>
</tbody>
</table>

### User

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the user</td>
</tr>
<tr>
<td>IP</td>
<td>String</td>
<td>The ip of the user</td>
</tr>
<tr>
<td>Global_Position</td>
<td>String</td>
<td>The package containing latitude and longitude of the user</td>
</tr>
</tbody>
</table>

### Vehicle

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the vehicle</td>
</tr>
<tr>
<td>Path</td>
<td>String</td>
<td>The path which the vehicle follows</td>
</tr>
<tr>
<td>Cost</td>
<td>Double</td>
<td>The cost of the vehicle</td>
</tr>
<tr>
<td>Time</td>
<td>Time</td>
<td>The time spent for the travel</td>
</tr>
</tbody>
</table>

### Vehicle type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the vehicle type</td>
</tr>
<tr>
<td>Type_Name</td>
<td>String</td>
<td>The name of the vehicle type</td>
</tr>
<tr>
<td>Cost_Multiplier</td>
<td>Double</td>
<td>The value showing the cost for a vehicle in a unit distance</td>
</tr>
<tr>
<td>Time_Divider</td>
<td>Double</td>
<td>The value showing the time for a vehicle in a unit distance</td>
</tr>
</tbody>
</table>

### Logs

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_ID (key)</td>
<td>Integer</td>
<td>Uniquely defines the logs</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
<td>The date of the log</td>
</tr>
</tbody>
</table>

### Modifies

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ID (key)</td>
<td>Integer</td>
<td>References : Staff</td>
</tr>
<tr>
<td>A_ID(key)</td>
<td>Integer</td>
<td>References : Activity</td>
</tr>
</tbody>
</table>

### Modifies 2

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ID (key)</td>
<td>Integer</td>
<td>References : Staff</td>
</tr>
<tr>
<td>A_ID(key)</td>
<td>Integer</td>
<td>References : Activity</td>
</tr>
</tbody>
</table>
### Works_In

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ID (key)</td>
<td>Integer</td>
<td>References : Staff</td>
</tr>
<tr>
<td>AP_ID(key)</td>
<td>Integer</td>
<td>References : Activity Places</td>
</tr>
</tbody>
</table>

### On

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ID (key)</td>
<td>Integer</td>
<td>References : Activity</td>
</tr>
<tr>
<td>AP_ID(key)</td>
<td>Integer</td>
<td>References : Activity Places</td>
</tr>
</tbody>
</table>

### Has_Type

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_ID (key)</td>
<td>Integer</td>
<td>References : Activity</td>
</tr>
<tr>
<td>T_ID(key)</td>
<td>Integer</td>
<td>References : Type</td>
</tr>
</tbody>
</table>

### Searches

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_ID (key)</td>
<td>Integer</td>
<td>References : User</td>
</tr>
<tr>
<td>A_ID(key)</td>
<td>Integer</td>
<td>References : Activity</td>
</tr>
<tr>
<td>L_ID(key)</td>
<td>Integer</td>
<td>References : Logs</td>
</tr>
</tbody>
</table>

### Occurs_In

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP_ID (key)</td>
<td>Integer</td>
<td>References : Activity Places</td>
</tr>
<tr>
<td>M_ID(key)</td>
<td>Integer</td>
<td>References : Map</td>
</tr>
</tbody>
</table>

### Occurs_In_2

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N_ID (key)</td>
<td>Integer</td>
<td>References : Node</td>
</tr>
<tr>
<td>M_ID(key)</td>
<td>Integer</td>
<td>References : Map</td>
</tr>
</tbody>
</table>

### Connected_To

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1_ID (key)</td>
<td>Integer</td>
<td>References : Node</td>
</tr>
<tr>
<td>N2_ID(key)</td>
<td>Integer</td>
<td>References : Node</td>
</tr>
<tr>
<td>Distance</td>
<td>Double</td>
<td>The distance between nodes</td>
</tr>
</tbody>
</table>

### Passes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_ID (key)</td>
<td>Integer</td>
<td>References : Vehicle</td>
</tr>
<tr>
<td>N_ID(key)</td>
<td>Integer</td>
<td>References : Node</td>
</tr>
</tbody>
</table>
### 3.2. Functional Model

#### 3.2.1. Data Flow Diagram

#### 3.2.1.1. DFD Level 0

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_ID (key)</td>
<td>Integer</td>
<td>References : Vehicle</td>
</tr>
<tr>
<td>VT_ID(key)</td>
<td>Integer</td>
<td>References : Vehicle Type</td>
</tr>
</tbody>
</table>
3.2.1.2. DFD Level 1

LEVEL 1 DIAGRAM

ADMINISTRATOR

CONTENT MANAGER

USER

STAFF DATABASE

ADMIN INTERFACE 1.0

CONTENT INTERFACE 7.0

USER INTERFACE 2.0

PROCESS STAFF QUERY 3.0

HANDLE XML 4.0

MANIPULATE GIS DATA 5.0

GPS DATA

DATABASE
3.2.1.3. DFD Level 2 (1.0)

LEVEL 2 DIAGRAM (1.0 - ADMIN INTERFACE)
3.2.1.4. DFD Level 2 (2.0)
3.2.1.5. DFD Level 2 (3.0)

```
LEVEL 2 DIAGRAM (3.0 - PROCESS STAFF QUERY)

Admin Information → SEND XML HANDLER (DECOMPOSE) 3.1 → Initial Staff Query (XML)

CM Information

Initial Staff Query → RETRIEVE FROM DATABASE 3.2 → Staff Query

Staff Query Result → SEND XML HANDLER (COMPOSE) 3.3 → Final Staff Query Result

Final Staff Query Result (XML) → SEND TO INTERFACES FOR DISPLAY 3.4

Display Screen

Display Screen CM
```
3.2.1.6. DFD Level 2 (4.0)

LEVEL 2 DIAGRAM (4.0 - HANDLE XML)

3.2.1.7. DFD Level 2 (5.0)

LEVEL 2 DIAGRAM (5.0 - MANIPULATE GIS DATA)
3.2.1.8. DFD Level 2 (6.0)

LEVEL 2 DIAGRAM (6.0 - PROCESS USER QUERY)

- GPS Data & Queries II
- User Query
- Build Map
- Node List
- Map Data
- Map Query
- Log Query
- Node List Copy
- Initial Node List
- Manipulated Queries
- Result List
- Result List
- User Query
- CREATE SQL QUERIES 6.2
- MANAGE ONTOLOGY 6.1
- GENERATE NODE & RESULT LIST 6.3
- DOUBLE NODE LIST 6.4
- RETRIEVE MAP 6.5

CoreAccess

Detailed Design Report

CoreTech
3.2.1.9. DFD Level 2 (7.0)

LEVEL 2 DIAGRAM (7.0 - CM INTERFACE)

- UNAME/Passw0rd
- UserName Password CM
- Response CM
- Login Info CM
- Activity Process
- CM Information
- Generate CM Information 7.3
- Display Screen CM
- Login Status CM

CHECK LOGIN CM 7.1
- GENERATE LOGIN STATUS CM 7.2
- FINAL DISPLAY CM 7.4
3.2.2. Data Dictionaries

<table>
<thead>
<tr>
<th>Name:</th>
<th>Login UserID/Passwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Admin Username – Password</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>ADMINISTRATOR output</td>
</tr>
<tr>
<td></td>
<td>CHECK LOGIN (1.1) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends administrator’s username and password for checking”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>UserName Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Admin Username- Password</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>CHECK LOGIN (1.1) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“checks username and password of the admin from staff database”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Check Result</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>CHECK LOGIN (1.1) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends the info about the username password match from database ”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Login Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Login Info After Check</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GENERATE LOGIN STATUS(1.2) input</td>
</tr>
<tr>
<td></td>
<td>CHECK LOGIN (1.1) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“forms the login information after login check”</td>
</tr>
<tr>
<td>Name: Administrative commands</td>
<td>Alias: System commands</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Where &amp; How it is used: ADMINISTRATOR output GENERATE ADMIN INFORMATION(1.3) input</td>
<td></td>
</tr>
<tr>
<td>Description: “written commands to interface by administrator”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Login Status Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias: -</td>
</tr>
<tr>
<td>Where &amp; How it is used: GENERATE LOGIN STATUS(1.2) output FINAL DISPLAY (1.4) input</td>
</tr>
<tr>
<td>Description: “sends the final login info for display”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Login Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias: User check</td>
</tr>
<tr>
<td>Where &amp; How it is used: ADMINISTRATOR input FINAL DISPLAY (1.4) output</td>
</tr>
<tr>
<td>Description: “information about validity of the user”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Administrative Commands Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias: System Returned Result</td>
</tr>
<tr>
<td>Where &amp; How it is used: ADMINISTRATOR input FINAL DISPLAY (1.4) output</td>
</tr>
<tr>
<td>Description: “returned information to the administrator whether the changes are done or not”</td>
</tr>
<tr>
<td>Name:</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Alias:</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Description:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Display Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Visible result</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>SEND TO INTERFACES FOR DISPLAY (3.4) output</td>
</tr>
<tr>
<td></td>
<td>FINAL DISPLAY (1.4) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“generated result screen to the administrator”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Result List(XML)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Result screen</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>COMPOSE QUERY (4.2) output</td>
</tr>
<tr>
<td></td>
<td>LIST THE RESULTS (2.5) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“generated result screen to the user about his/her query”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>GPS Data &amp; Queries I(XML)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Written data</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>DECOMPOSE QUERY(4.1) input</td>
</tr>
<tr>
<td></td>
<td>FORM XML QUERY (2.4) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“Gathered info from user interface”</td>
</tr>
<tr>
<td>Name</td>
<td>Request Info</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Where &amp; How it is used</td>
<td>USER output</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends user request for any activity”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>User Request</th>
<th>Alias:</th>
<th>GPS Data Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where &amp; How it is used:</td>
<td>GET GPS DATA (2.1) output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>“sends GPS data request to GPS receiver”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>GPS Data</th>
<th>Alias:</th>
<th>GPS Data Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where &amp; How it is used:</td>
<td>GET GPS DATA (2.1) input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>“sends GPS data from GPS receiver”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>User Request M</th>
<th>Alias:</th>
<th>GPS Data Request Manually</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where &amp; How it is used:</td>
<td>GET GPS DATA (2.1) output</td>
<td>INSERT GPS MANUALLY (2.2) input</td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>“sends GPS data request for manual insertion”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name:</td>
<td>GPS Data M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alias:</td>
<td>GPS Data Response Manually</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GET GPS DATA (2.1) input</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSERT GPS MANUALLY (2.2) output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description:</td>
<td>“sends GPS data from manual insertion”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Info with GPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Data with GPS</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GET GPS DATA (2.1) output</td>
</tr>
<tr>
<td></td>
<td>SEARCH ACTIVITY (2.3) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends info for other processes with GPS Data ”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Info with Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Data with Activity</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>SEARCH ACTIVITY (2.3) output</td>
</tr>
<tr>
<td></td>
<td>FORM XML QUERY (2.4) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends activity info for other processes”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Select Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Select Item from Result List</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>LIST THE RESULTS (2.5) output</td>
</tr>
<tr>
<td></td>
<td>DISPLAY SELECTED WITH DETAILS (2.6) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“selects item from result list for further details ”</td>
</tr>
<tr>
<td>Name:</td>
<td>Result Map</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>Alias:</td>
<td>Final Map Data</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>USER input</td>
</tr>
<tr>
<td></td>
<td>DISPLAY SELECTED WITH DETAILS (2.6) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“shows the final map to user”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>CM Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Content Manager Data</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GENERATE CM INFORMATION (7.3) output</td>
</tr>
<tr>
<td></td>
<td>SEND XML HANDLER (DECOMPOSE) (3.1) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“gathered content manager info”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Initial Staff Query (XML)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>XML Query for Decomposition</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>SEND XML HANDLER (DECOMPOSE) (3.1) output</td>
</tr>
<tr>
<td></td>
<td>DECOMPOSE QUERY (4.1) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends staff query in XML format for decomposition”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Initial Staff Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Query Response after Decomposition</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>DECOMPOSE QUERY (4.1) output</td>
</tr>
<tr>
<td></td>
<td>RETRIEVE FROM DATABASE (3.2) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“receives staff query after decomposition”</td>
</tr>
<tr>
<td>Name</td>
<td>Staff Query</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Alias</td>
<td>-</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>RETRIEVE FROM DATABASE (3.2) output</td>
</tr>
<tr>
<td>Description</td>
<td>“sends staff query to database”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Staff Query Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>Staff Query from Database</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>SEND XML HANDLER (COMPOSE) (3.3) input</td>
</tr>
<tr>
<td>Description</td>
<td>“receives staff query results from database”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Final Staff Query Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>Query for XML Composition</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>SEND XML HANDLER (COMPOSE) (3.3) output</td>
</tr>
<tr>
<td></td>
<td>COMPOSE QUERY (4.2) input</td>
</tr>
<tr>
<td>Description</td>
<td>“sends final staff query for XML composition”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Final Staff Query Result (XML)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>Query Response after Composition</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>COMPOSE QUERY (4.2) output</td>
</tr>
<tr>
<td></td>
<td>SEND TO INTERFACES FOR DISPLAY(3.4) input</td>
</tr>
<tr>
<td>Description</td>
<td>“receives final staff query after decomposition”</td>
</tr>
<tr>
<td>Name:</td>
<td>Display Screen CM</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Alias:</td>
<td>Visible result for CM</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>SEND TO INTERFACES FOR DISPLAY (3.4) output</td>
</tr>
<tr>
<td></td>
<td>FINAL DISPLAY CM (7.4) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“generated result screen to the content manager”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>GPS Data &amp; Queries II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Query Info after Decomposition</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>DECOMPOSE QUERY (4.1) output</td>
</tr>
<tr>
<td></td>
<td>MANAGE ONTOLOGY (6.1) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends the user query for further process”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Manipulated Map Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Processed Map Data</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>DRAW PATH (5.3) output</td>
</tr>
<tr>
<td></td>
<td>COMPOSE QUERY (4.2) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends the manipulated map data for XML composition and display”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Result List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Result List before XML Composition</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GENERATE NODE &amp; RESULT LIST (6.3) output</td>
</tr>
<tr>
<td></td>
<td>COMPOSE QUERY (4.2) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends the result list for XML composition”</td>
</tr>
<tr>
<td>Name:</td>
<td>Node List</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Alias:</td>
<td>Nodes for Map</td>
</tr>
<tr>
<td>Where &amp; How</td>
<td>DOUBLE NODE LIST (6.4) output</td>
</tr>
<tr>
<td>it is used:</td>
<td>FIND SHORTEST PATH (5.2) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends node information for map manipulation”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Map Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Extracted Map</td>
</tr>
<tr>
<td>Where &amp; How</td>
<td>BUILD MAP (6.6) output</td>
</tr>
<tr>
<td>it is used:</td>
<td>DOUBLE MAP DATA (5.1) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends the received map data from database”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Map Data I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>-</td>
</tr>
<tr>
<td>Where &amp; How</td>
<td>DOUBLE MAP DATA (5.1) output</td>
</tr>
<tr>
<td>it is used:</td>
<td>FIND SHORTEST PATH (5.2) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends a copy of map data for finding shortest path”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Map Data II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>-</td>
</tr>
<tr>
<td>Where &amp; How</td>
<td>DOUBLE MAP DATA (5.1) output</td>
</tr>
<tr>
<td>it is used:</td>
<td>DRAW PATH (5.3) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends a copy of map data for drawing path”</td>
</tr>
<tr>
<td>Name:</td>
<td>Path Coordinates</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Alias:</td>
<td>Positions of Path</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>FIND SHORTEST PATH (5.2) output</td>
</tr>
<tr>
<td></td>
<td>DRAW PATH (5.3) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends path coordinates for drawing path”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Manipulated Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Processed Queries</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>MANAGE ONTOLOGY (6.1) output</td>
</tr>
<tr>
<td></td>
<td>CREATE SQL QUERIES (6.2) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends the manipulated queries to form SQL queries”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>User Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>-</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>CREATE SQL QUERIES (6.2) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends user query to database”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Log Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>-</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>CREATE SQL QUERIES (6.2) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“writes the logs to the database”</td>
</tr>
<tr>
<td>Name:</td>
<td>User Query Result</td>
</tr>
<tr>
<td>-------</td>
<td>------------------</td>
</tr>
<tr>
<td>Alias:</td>
<td>Result from Database</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GENERATE NODE &amp; RESULT LIST (6.3) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“receives user query result from database”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Initial Node List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Map Nodes</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GENERATE NODE &amp; RESULT LIST (6.3) output DOUBLE NODE LIST (6.4) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends map nodes for making two copies”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Node List Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>-</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>DOUBLE NODE LIST (6.4) output RETRIEVE MAP (6.5) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends map nodes for retrieving map”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Map Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Map Retrieving</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>RETRIEVE MAP (6.5) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“sends map query to database”</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Map Query Result</td>
<td>“sends the map data from database for building map”</td>
</tr>
<tr>
<td>Uname/Passwd</td>
<td>“sends content manager’s username and password for checking”</td>
</tr>
<tr>
<td>UserName Password CM</td>
<td>“checks username and password of the content manager from staff database”</td>
</tr>
<tr>
<td>Response CM</td>
<td>“sends the info about the username password match from database”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map from Database</td>
<td></td>
</tr>
<tr>
<td>Content Manager Username – Password</td>
<td></td>
</tr>
<tr>
<td>Check Result for CM</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Where &amp; How it is used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD MAP (6.6) input</td>
</tr>
<tr>
<td>CONTENT MANAGER output</td>
</tr>
<tr>
<td>CHECK LOGIN CM(7.1) input</td>
</tr>
<tr>
<td>CHECK LOGIN CM (7.1) output</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>“sends the map data from database for building map”</td>
</tr>
<tr>
<td>“sends content manager’s username and password for checking”</td>
</tr>
<tr>
<td>“checks username and password of the content manager from staff database”</td>
</tr>
<tr>
<td>“sends the info about the username password match from database”</td>
</tr>
<tr>
<td>Name:</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>Alias:</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Description:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Activity Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>Activity Work</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>CONTENT MANAGER output</td>
</tr>
<tr>
<td></td>
<td>GENERATE CM INFORMATION(7.3) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“written commands to interface by content manager”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Login Status Display CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>-</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>GENERATE LOGIN STATUS CM(7.2) output</td>
</tr>
<tr>
<td></td>
<td>FINAL DISPLAY CM (7.4) input</td>
</tr>
<tr>
<td>Description:</td>
<td>“ sends the final login info for display”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name:</th>
<th>Login Status CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>User check for CM</td>
</tr>
<tr>
<td>Where &amp; How it is used:</td>
<td>CONTENT MANAGER input</td>
</tr>
<tr>
<td></td>
<td>FINAL DISPLAY CM (7.4) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“information about validity of the user ”</td>
</tr>
</tbody>
</table>
### Name: Process Results

<table>
<thead>
<tr>
<th>Alias:</th>
<th>Activity Process Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where &amp; How it is used:</td>
<td>CONTENT MANAGER input</td>
</tr>
<tr>
<td></td>
<td>FINAL DISPLAY CM (7.4) output</td>
</tr>
<tr>
<td>Description:</td>
<td>“returned information to the content manager whether the changes are done or not”</td>
</tr>
</tbody>
</table>

## 4. OVERALL ARCHITECTURE

### 4.1. List of modules

- GPS module
- PDA module
- Content Manager module
- Administrator module
- Web Service module
- Data Object Handler
- Logger module
- GIS Engine
- Ontology module
- Pathfinder module
- Activities module
- Map module
- Transportation module
- Database module
4.2. Architecture Diagram
4.3. Details of Modules

Main details of the modules will be explained in the following part. However, detailed functionalities of the modules are clear in the diagrams, especially the interaction between them.

**GPS Module:** GPS module’s responsibility is to make the connection between the PDA and GPS receiver. Moreover, it is responsible of parsing the strings of NMEA 0183 protocol and returning them to PDA module as a data object which consists of global position information such as latitude, longitude and time attributes of the user. NMEA 0183 protocol serves lots of different kinds of strings, but “recommended minimum” sentence, namely $GPRMC, meets our necessities very well. Detailed information about the contents of GPS module can be found in our “GPS Research Report” which is accessible in our website [www.cclub.metu.edu.tr/~mustafa/coretech](http://www.cclub.metu.edu.tr/~mustafa/coretech).

**PDA Module:** PDA module’s responsibility is to supply the connection between the user and web service module of server side. It has a friendly graphical user interface which takes in the commands of the user and displays the related outputs. Basic operations such as zooming in/out the vector map are inline facilities of PDA module and connection with the server is not necessary in these cases. All of the other requests of user need connection with the server side. Requests of the user are taken through the GUI of PDA module, then packaged with XML Handler component as XML and sent to the server side via the web service interface of the server application. Next, it takes the response of the server again in XML format, decomposes it and displays to user as maps and written data.

**Content Manager Module:** Content manager module’s responsibility is to supply the connection between the content manager and web service module of server side. It has a graphical user interface which takes in the commands of the content manager and returns the acknowledgement of the operation to content manager. As explained before, content manager has restricted abilities. He/She is responsible of managing the activities of activity places where he/she is assigned. Moreover, he/she can update contact information of activity places. All these operations need connection with the server side. This connection is made through the web service interface of the server side. Messaging standard is XML again, just as explained in the PDA module description. Since content managers will want to do their jobs from anywhere, application of this module will be accessible as a web page which connects with the web services.

**Administrator Module:** Administrator module is very critical since it deals with the administration of the server side. It provides the administrator with a friendly GUI which can be seen in the GUI Design part of this report. It acts in a similar way with the content manager module; it supplies the connection between the administrator and web service interface of the server side. Communication protocol is again XML. However, this time, application will not be accessible as a web page because administrator has some complex abilities such as managing the map visually. Therefore, this module will be a standalone application which makes the connection itself. Authentication will be done of course at the beginning.

**Web Service Module:** Web service module is the open window of our server application to outside world. All of the capabilities of our server side will be deployed as web services. This will bring us the platform independency. Authentication will be done again in web service module for the administrator and content manager. Once our WSDL is parsed by the clients,
our server side will be available just like an API as long as they stick to the data specification standards of our web services. To sum up, it can be said that this module is the interface of server side for clients.

**Data Object Handler:** Data object handler works in a two-sided way. Its first duty is to take the requests of clients (namely, PDA module, client manager module and administrator module) via web service module, decompose them and create understandable data objects for the GIS Engine. Secondly, it will just do the reverse operation. That is, it will take the data objects of the GIS Engine which consist of the responses of the server and create the package for the client side and pass it to the clients (again PDA, client manager and administrator modules) via web service module.

**Logger Module:** Logger module lies between the data object handler and database module. Its duty is to log the queries of the user with his/her global position and ip. Logger module has its own table in the database. Database module handles the process of entering the logs that has been constructed and sent by logger module to logger table. These logs will serve statistical data of the whole application to administrator.

**GIS Engine:** The core module of our system is the GIS engine which sits in the middle and controls the data flow mechanism. As it can be clearly seen from the diagram, data object handler, logger module, ontology module, pathfinder module, activities module, map module and transportation module are connected to GIS engine. GIS engine takes the data objects from the data object handler, and then according to the data, it communicates with the related module when necessary. For example, if the user requested transportation alternatives to an activity place, then GIS engine sends the request to the transportation module, gets the response and sends it back to the user. It is the brain of our project.

**Ontology Module:** Main purpose of this module is to prevent the user from facing with blank screens instead of alternative solutions. Ontology module is only connected with GIS engine. If the response for the original query of the user is empty, then GIS engine will call ontology module and make it produce alternative results. For example, if the user wanted to see a “love movie” but there is not any on the scene, then “love comedy movie” or “love play” would be alternative recommendations.

**Pathfinder Module:** This module is in connection with GIS engine and transportation module. Its main duty is to find the paths between given points. In fact, its duty can be divided into two parts: obligatory and optional. Its obligatory duty is to find the general shortest path between the users’ global position and found activity place. This shortest path is transportation method (vehicle) independent. Pathfinder module’s optional duty gets into action when user wants to see transportation alternatives for an activity place. This time, pathfinder works for specific vehicles. Behind, there are implementations of shortest path and graph algorithms.

**Activities Module:** This module is responsible of managing the activities and activity places. GIS engine redirects the data objects that are concerned with activities to this module. Then, activities module gets into connection with database module, sends the queries and gets back the responses. Finally, it sends these results back to GIS engine. Messaging is done via objects and strings. For the content manager and administrator module, it has extra properties such as adding, updating and removing activities or activity places.
Map Module: Map module is just like the activities module. It lies between the GIS engine and database module. According to the request taken from the GIS engine, map module orders related map data from the database module and sends the result back to GIS engine. For the administrator and content manager, there are extra methods for editing the map data.

Transportation Module: Transportation method is responsible for the vehicles and their paths. Managing vehicles and their paths are only visible to administrator. User queries are taken from the GIS engine and sent to database module. Database module takes the response from the SQL tables and passes them to transportation module. Transportation module’s duty is completed when responses are sent back to GIS engine.

Database Module: As it is clear from its name, database module is responsible of the connection between the database management system (MySQL in our case) and other modules (logger, activities, map and transportation modules) that need database access. Database module takes the queries from the mentioned modules, commits them via the database connection, gets back the result set and forwards the related results to interested modules.
5. COREACCESS USER INTERFACE

5.1. User Functionality

We have two different users for “CoreAccess”. Main users are PDA users who run application from a PDA connected to GPS and internet. However, some PDA users may not have GPS receivers. For this reason, we have added extra positioning methods which are explained in detail in the following part. Second group of users are internet users. Since our server application will serve the information via web services, this will not bring us any additional load. All we have to is designing a web page which calls our web services, then internet users can also benefit from our GIS application. The only difference is, internet user does not have GPS receiver. There is not any other difference between the functionalities of PDA user and internet user. The functionalities of users are explained in detail in the following sections.

- Select Positioning Method: This is the first step of our application. Before making any queries, we have to know the position of the user. User has four choices for determining his/her position. First one is determining user’s global position “via GPS receiver”, second one is “via entering address keyword”, third one is “via browsing on map” and the last one is “via manually entering global position”. We serve any kind of possibilities for positioning, because main purpose of our application depends on global position of user. In some cases user may not have GPS receiver, so other alternatives are important also.

![Select Positioning Method](image)
The details of these methods are:

- **Via GPS Receiver**: This is the first option. If user has a GPS receiver connected to PDA, then this option is enabled. GPS serves the most accurate positioning information among our other positioning methods. The only thing that user has to do is selecting this option. Then NMEA 01803 strings will be read from the serial port, $GPRMC sentences, which are the “recommended minimum” sentences that contain longitude, latitude and altitude information (detailed information can be found in our GPS research report), will be interpreted by our GPS string interpreter class. Once this option is selected, user’s position will be updated automatically in specific time intervals like 10 seconds. Additionally, this is the only option that web users are unable to select.

- **Via Address Keyword**: This option is added by the advice of our instructors and we believe it is very crucial. When user does not have a GPS receiver, (s)he can enter some keywords which may be consisting of streets, avenues, towns, etc. Then search query is sent to server and vector map of that place is shown to user. For example, user may enter “100.yıl Pazarı” or “Necatibey Caddesi”, then a map with its center having the found place is returned to user. User can now easily determine his/her exact position by browsing on the map, navigating, zooming in and out. All properties of map menu are available to user at that time. This is the best option for web users also.

- **Via Browsing on Map**: In fact, this option is similar to the previous one. The only difference is, now user does not specify any address keyword. When this option is selected, city map of Ankara for example is displayed to user. Then the user is able to determine his/her exact position by browsing on the map, navigating, zooming in and out. Again, all properties of map menu are available to user at that time. This option is also beneficial for web users. When user decides a point on the map, the latitude and longitude information of that point is determined and sent to server as the global position of user.

- **Via Manually Entering Global Position**: This is the last option and its main purpose is to meet our testing demands when we don’t have GPS receiver, in fact. However there may be some users who are sure about their latitudes and longitudes especially by the help of observing Google Earth in these days. We can not claim that this option is helpful to users in general, so maybe we will hide this property in the final release of our product. Its working mechanism is very easy, user only fills in the latitude and longitude text fields. Then this information is sent to server as global position of the user.

- **Search Activity**: User’s global position is taken, now the search menu appears to the user. The application area of our project is social activities. These activities are cinema, theatre, music and sport. There are three different search options in this menu. First one is “search by category” option, second is “search by attributes” option and the last one is “search by browsing map” option. The results of all these three options are identical, they will return a result list. This is very important for modular programming. Graphical representation of this situation can be found in our early State Transition Diagrams. This result list will contain the major identities of found places, which will be explained in the following part.
The details of these three menus are:

- **Search by category**: If user wants to do an activity, but no matter the place, he/she can search for the appropriate places. The only necessity is to determine a category name between the cinema, theatre, music and sport categories. In fact, this option is for users who do not have any idea about what to do. For this reason, user may want to see the social activities around him. For instance, if a user in METU campus decides to go to cinema and select “cinema” category in this menu, our application will find the nearest cinemas around him. Results for this example may be first METU cinema (U3), then Bilkent and finally Tüze Armada. Moreover, he/she can view from the map.
Search by attributes: This option is for users who have at least some idea about what to do. User is able to enter his criteria for the activity he/she wants to do. These criteria may be:

- Activity name,
- Activity place,
- Activity category,
- Let activity date between preferred dates.

User is able to specify none, one or many of these criteria. The number of entered criteria increases the detail level of search. Then, according to entered criteria, search is successfully done and results are shown to user as a list sorted according to smallest distance value.
Search by browsing map: This option is for users who do not want to use text based menus for searching. In this menu, according to users’ global position, local map is shown to user. Moreover, users can also enter some address keywords and then system will bring the related piece of map. User position is signed on the map. He can browse, zoom in/out, rotate map and select the activity place. Visible activity places may vary according to zoom level of the map. After deciding on the activity place from the map, following options will be the same as previous search methods.
- **Select from Result List**: All three search options mentioned above come to this menu. The results are displayed to user as a list sorted from nearest to furthest. There may be more than one result, on the other side search may return empty list if even our ontology definitions fail to find a result. The activities and places are listed with their keywords in this menu. User can select one or more items from this menu. Next, he has two options on the selected items. User can either see the details of the activity and place in written form or see the places on the map. The details of these menus are explained in the following sentences.
• **Display Written Details:** This menu shows the details of the selected activity or place. The written details in this menu include:

  o Activity name,
  o Activity place,
  o Activity date and time,
  o Activity place’s address and phone,
  o Extra properties of the place like having parking place, children playground, etc,
  o Link to transportation options.

User can easily go back to returned results menu and display another item’s details. Transportation options are not shown automatically because user may not want to see them. Therefore, there is a link for transportation options in this menu. If user selects it, then transportation menu will appear.

• **Transportation Menu:** This menu is a sub menu of “display written details” menu because it is an optional menu. If user selects it, the transportation options between the user’s global position and activity place’s global position will be shown to user. This menu has both written part and visual part which consists of the shortest path displayed on vector map. The written part includes the followings for the selected transportation option:

  o Names of the vehicles (since there may be combination of vehicles as a result),
- Vehicle information (for ex: 132: ODTU – Kızılay),
- Estimated distance,
- Estimated time (for the work hours, this variable may be treated differently),
- Cost of the vehicle combination.

The transportation methods mentioned here may be “bus”, “dolmuş”, “metro”, “tramway”, “taxi” and finally “on foot”. We have the path of each transportation method in our database. There may be cases when there is more than one possible way to reach activity place, or there may not be any public transportation vehicles available at that time to desired place and taxi would be the only solution. For all cases, “taxi” is the final option in CoreAccess.

Secondly, as we have mentioned above, user is able to see the combination of vehicles on the map. This time, shortest path for the selected combination will be drawn on vector map. For finding shortest paths, we will use efficient shortest path and graph algorithms. Again, user is able to do all the functionalities of map menu like zooming in / out, navigating, etc. Moreover, the distance of the paths may be shown on the map. This is a good option for comparing the distances of the paths. With the help of this property, user can manage his time efficiently. In the same manner as previous “Display Distance” option, estimated time and cost values may be shown on the map also. This property will increase the time and cost efficiency of the user’s choice.
- **Display Items on Map**: This menu is second sub menu of “result list” menu. In this menu user is able to see his selections on the map. User may select more than one place in the previous selection menu. On the map, his global position and selected activity places will be shown with a placemark. The paths directed from his global position to activity places will also be drawn in different colors. Furthermore, user can make following operations on the map:
  
  - **Zoom in/out**: Map can be zoomed in / out.
  - **Rotate**: Map can be rotated either clockwise or counter clockwise.
  - **Browse Map**: User does not have to stick to the result map. He is able to browse the map by going upwards, downwards, left and right.

User can easily go back to returned results menu and display another item’s details on map.
• Select Language: Our user application is multilingual. This is a very easy thing to do in fact, we don’t expect a locale problem.

![Select Language](image)

5.2. Content Manager Functionality

The duty of content manager is to change the contents of their company’s social activities. For instance, the content manager of a cinema can add the new films which are on screen. Content manager may be responsible of managing several places’ activities. The followings are the capabilities of the content manager’s functions:

• Login: A content manager has to login to the system first with his id and password. A content manager may have the ability to modify only one place’s activities or a number of places’ activities.

• Add/Modify/Delete Activity: If new activities are available, content manager has the responsibility to add new activities and move the past activities to history. There may be cases when content manager needs to delete or modify the activity. The properties of the activities need to be entered are:
  
  o Activity Name
  o Activity Time
  o Activity Place (hall)
- Activity Cost
- Activity Type

Managing Activities

- **Add/Modify short description about the activity:** Content manager can add brief information about the activity. Thus, users can have some idea before attending any activity.

- **Add/Modify phone number/address/e-mail address:** Content manager can add or modify those important attributes of the activity place.
5.3. **System Administrator Functionality**

System Administrator has the highest level rights. He is able to do anything that user and content manager can do. Apart from those rights, he is responsible for uploading map data, ontology information, determining the relation between instances in the ontology. Functionalities of system administrator are explained below extensively:

- **Login:** In order to accomplish main activities, system administrator has to login to the system for security. This feature enables the protection of database contents of the application. Only system administrator has the right to modify and add the contents of the ontology and map data.

- **Upload/Delete/Modify Map:** In this functionality, system administrator can upload map, delete map, modify map. Some new areas that were not in the coverage area of CoreAccess can be added to extend the coverage area. In the same manner, some areas may be discarded or modified.

- **Add/Modify/Delete Node:** Nodes are very important in CoreAccess. All maps are processed as connected nodes. Activity places are special nodes. Apart from activity places, there are lots of nodes to describe roads. Vehicles’ paths are constructed by series of connected nodes. As a result, its system administrator’s responsibility to manage all nodes.

- **Add/Modify/Delete Vehicle:** System administrator can add, modify or delete a vehicle. As explained in the previous part, vehicles’ paths are defined by connected nodes. System administrator can change the path of the vehicle by adding new nodes or
removing existing nodes. Moreover, system administrator can manage the type (which may be one of “bus”, “dolmuş”, “metro”, “tramway” or “taxi”), time_divider and cost_multiplier of the vehicle. time_divider and cost_multiplier are vehicle specific properties. These allow the system to estimate cost and duration of a path for selected transportation method.

Admin - Manage Vehicle

- **Define/Modify/Delete Ontology**: System administrator can define different ontologies for activities, especially for movies and plays.
• **Add/Modify/Delete Activity Places:** Content manager can manage activities but its system administrator’s responsibility to manage activity places. The attributes that have to be filled are name, address, phone number, e-mail address, Global Position and Node of the activity place.
• **Manage Content Managers:** Its system administrator’s responsibility to add new content managers or delete old ones. System administrator gives user id and password to content managers and defines their abilities. Therefore, content managers have limited right to access and modify the database elements.

![Admin - Manage Content Managers](image)

• **View Logs/History:** One of the most usable properties of CoreAccess is logging. CoreAccess logs all of the user activities. When user makes a search or requests details of any activity/place, CoreAccess stores this information to database. The attributes of the logs are: user ip, global position of the user, time and date, search details, requested activity’s details, etc. System administrator is able to see and print these logs any time. By this way, statistical data will be taken and managers and companies will be informed about the usage statistics.
- **Help Menu**: All applications should have a help menu, so this is for the administrator.
6. UML DIAGRAMS

6.1. Use Case Diagrams

6.1.1. Use Case Diagrams of the User

6.1.1.1. Positioning Use Case

**Flow of Events for the Positioning Use-case**

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the user to get GPS Data from 3 ways</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
<tr>
<td><strong>Main Flow</strong></td>
<td>1. User has 3 ways to get GPS Data.</td>
</tr>
<tr>
<td></td>
<td>2. First option is simply getting the GPS Data from a GPS Receiver connected to PDA.</td>
</tr>
<tr>
<td></td>
<td>3. Second option is browsing the map to find the location of the user.</td>
</tr>
<tr>
<td></td>
<td>4. Last option is mainly for users who know exact position of himself/herself by entering manually.</td>
</tr>
<tr>
<td>Post-condition</td>
<td>The user has the GPS information now. The system knows where the user is.</td>
</tr>
</tbody>
</table>

6.1.1.2. Search by Attributes Use Case
### Flow of Events for the Search by Attributes Use-case

<table>
<thead>
<tr>
<th><strong>Objective</strong></th>
<th>To allow the user to search the activities with a specific attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition</strong></td>
<td>The GSP Data should be received.</td>
</tr>
<tr>
<td><strong>Main Flow</strong></td>
<td>1. User interacts with the activity attributes interface.</td>
</tr>
<tr>
<td></td>
<td>2. User can view all the attributes for a specific activity. Actually there are some fixed attributes such as “activity name”, “activity place”, “activity date”, etc.</td>
</tr>
<tr>
<td></td>
<td>3. User can make search for the attributes he/she desires.</td>
</tr>
<tr>
<td></td>
<td>4. User can add/update/delete attributes.</td>
</tr>
<tr>
<td></td>
<td>5. Some extra features can be selected in addition to attributes.</td>
</tr>
<tr>
<td><strong>Post-condition</strong></td>
<td>The user managed the search by activity attributes.</td>
</tr>
</tbody>
</table>

#### 6.1.1.3. Search by Category Use Case

![Activity Category Interface Diagram](image)

<table>
<thead>
<tr>
<th><strong>Objective</strong></th>
<th>To allow the user to search the activities with a category.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition</strong></td>
<td>The GSP Data should be received.</td>
</tr>
<tr>
<td><strong>Main Flow</strong></td>
<td>1. User interacts with activity category interface.</td>
</tr>
<tr>
<td></td>
<td>2. User can view the four current categories, namely “cinema”, “theatre”, “concert hall” and “sport center”.</td>
</tr>
<tr>
<td></td>
<td>3. User can select category among these four categories.</td>
</tr>
<tr>
<td></td>
<td>4. User can search the database for the selected category.</td>
</tr>
<tr>
<td></td>
<td>5. Some extra features can be selected in addition to activity category.</td>
</tr>
<tr>
<td><strong>Post-condition</strong></td>
<td>The user managed the search by activity categories.</td>
</tr>
</tbody>
</table>
6.1.1.4. Search on Map Use Case

Flow of Events for the Search by Category Use-case

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the user to search the activity place on map.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>The GSP Data should be received.</td>
</tr>
</tbody>
</table>
| Main Flow | 1. User interacts with search on map interface.  
2. User can view the map and mark the activity place on the map.  
3. User can enter a keyword for searching an address.  
4. User can update the position he/she has already done.  
5. User can browse the map according to his/her wish.  
6. On the map the zoom in and zoom out applications can be applied.  
7. On the map rotate application can be applied.  
8. On the map scale application can be applied. |
| Post-condition | The user managed to browse the map for a specific activity. |
6.1.1.5. Display Results Use Case

Flow of Events for Display Results by Category Use-case

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the user to view the search result list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>The result list for the searches should be returned.</td>
</tr>
<tr>
<td>Main Flow</td>
<td>1. User interacts with result list interface.</td>
</tr>
<tr>
<td></td>
<td>2. User is faced with results when he/she makes a query.</td>
</tr>
<tr>
<td></td>
<td>3. After the results are displayed, user chooses one item from the result list.</td>
</tr>
<tr>
<td></td>
<td>4. When an item is selected, either the written details of the selection item or the map display of it is seen on the screen.</td>
</tr>
<tr>
<td></td>
<td>5. If user chooses the written details of the item, he/she comes across with the written information about the results. It contains activity name, place, date, time, address and phone number of the activity place, estimated distance etc. In addition to this, transportation options can be seen if user desires. The transportation options include choosing the transportation vehicle, viewing the estimated time and cost for the distance. User can switch to map view from this view.</td>
</tr>
<tr>
<td></td>
<td>6. If user chooses the map display of the selected result item, he/she can observe the activity place and the routine on the map. User can do browse, zoom in, zoom out, rotate and scale operations on the map. Switching to written details of the selected item is possible as told above.</td>
</tr>
<tr>
<td>Post-condition</td>
<td>The user managed to see the query results.</td>
</tr>
</tbody>
</table>
6.1.2. Use Case Diagrams of the Administrator

6.1.2.1. Login Use Case

Flow of Events for Login Use-case

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the administrator to get into the system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
| Main Flow | 1. Administrator interacts with the login interface.
2. Administrator is requested to enter his/her username and password for getting into the system.
3. After entering his/her information, the validity of these is checked.
4. If the information is false, the administrator is simply rejected, warned and prompted to enter the information again.
5. If the information is true, the administrator is allowed to get into the system. |
| Post-condition | The administrator is in the system now. |

6.1.2.2. Manage Map Use Case

Flow of Events for Manage Map Use-case

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the administrator to manage the map.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>The administrator should be logged in.</td>
</tr>
</tbody>
</table>
Main Flow

1. Administrator interacts with the map managing interface.
2. Administrator can view all the map lists.
3. Administrator can add new map to the map list.
4. Administrator can update a specific map.
5. Updating the map can be done by adding/modifying/deleting nodes.
6. Administrator can delete a specific map from the map list.

Post-condition

The administrator did map managing work.

6.1.2.3. Manage Activity Place Use Case

Flow of Events for Manage Activity Place Use-case

Objective

To allow the administrator to manage activity place.

Precondition

The administrator should be logged in.

Main Flow

1. Administrator interacts with activity place interface.
2. Administrator can view the entire activity place list.
3. Administrator can add new activity places.
4. Administrator can update a specific activity place.
5. Administrator can delete a specific activity place.

Post-condition

The administrator is managed to do activity place operations.
6.1.2.4. Manage Vehicle Use Case

**Flow of Events for the Manage Vehicle Use-case**

**Objective**
To allow the administrator to manage the vehicles.

**Precondition**
The administrator should be logged in.

**Main Flow**
1. Administrator interacts with vehicle interface.
2. Administrator can view the entire vehicle list.
3. Administrator can add new vehicles.
4. Administrator can update a specific vehicle.
5. Administrator can delete a specific vehicle.

**Post-condition**
The administrator did vehicle managing work.

6.1.2.5. Manage Ontology Use Case

**Flow of Events for the Manage Ontology Use-case**

**Objective**
To allow the administrator to manage the ontologies.

**Precondition**
The administrator should be logged in.

**Main Flow**
1. Administrator interacts with ontology interface.
2. Administrator can view the entire ontology list.
3. Administrator can add new ontologies.
4. Administrator can update a specific ontology.
5. Administrator can delete a specific ontology.

**Post-condition**
The administrator did ontology managing work.
6.1.2.6. Manage Content Managers Use Case

Flow of Events for the Manage Content Managers Use-case

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the administrator to manage the content managers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>The administrator should be logged in.</td>
</tr>
</tbody>
</table>
| Main Flow | 1. Administrator interacts with content manager interface.  
            2. Administrator can view the entire content manager list.  
            3. Administrator can add new content managers.  
            4. Administrator can update a specific content manager.  
            5. Administrator can delete a specific content manager.  
            6. Administrator can view the content manager statistics. |
| Post-condition | The administrator did content manager managing work. |

6.1.2.7. Log Use Case

AUTHENTICATED ADMIN
# Flow of Events for the Log Use-case

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the administrator to manage the logs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>The administrator should be logged in.</td>
</tr>
</tbody>
</table>
| **Main Flow** | 1. Administrator interacts with log interface.  
2. Administrator can monitor the relevant user log.  
3. Administrator can monitor the activity logs.  
4. Administrator can monitor activity place log.  
5. Administrator can monitor category logs. |
| **Post-condition** | The administrator did log managing work. |

---

## 6.1.3. Use Case Diagrams of the Content Manager

### 6.1.3.1. Login Use Case

![Login Use Case Diagram]

---

# Flow of Events for Login Use-case

<table>
<thead>
<tr>
<th>Objective</th>
<th>To allow the content manager to get into the system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>-</td>
</tr>
</tbody>
</table>
| **Main Flow** | 1. Content manager interacts with the login interface.  
2. Content manager is requested to enter his/her username and password for getting into the system.  
3. After entering his/her information, the validity of these is checked.  
4. If the information is false, the content manager is simply rejected, warned and prompted to enter the information again.  
5. If the information is true, the content manager is allowed to get into the system. |
| **Post-condition** | The content manager is in the system now. |
6.1.3.2. Manage Activity Use Case

**Flow of Events for Manage Activity Use-case**

**Objective**
To allow the content manager to manage the activities.

**Precondition**
The content manager should be logged in.

**Main Flow**
1. Content manager interacts with the activity interface.
2. Content manager can view the entire activity list.
3. Content manager can add a new activity to the list.
4. Content manager can update a specific activity.
5. Updating an activity can be done by adding/modifying/deleting the activity description.

**Post-condition**
The content manager managed to do activity work.

6.1.3.3. Manage Activity Place Use Case

**Flow of Events for Manage Activity Place Use-case**

**Objective**
To allow the content manager to manage the activity places.

**Precondition**
The content manager should be logged in.
<table>
<thead>
<tr>
<th>Main Flow</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Content manager interacts with the activity places interface.</td>
<td>2. Content manager can update the activity places.</td>
</tr>
<tr>
<td>3. Content manager can update the phone numbers of activity places.</td>
<td>4. Content manager can update the addresses of activity places.</td>
</tr>
<tr>
<td>5. Content manager can update the e-mails of activity places.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The content manager managed to do activity places work.</td>
</tr>
</tbody>
</table>
6.2. Class Diagrams

6.2.1. PDA Class Diagram
The classes in this part are PDA Gui and some interaction classes which communicates via web service with server. In order to manage user interface forms a class called **PDA Gui** is designed. This class contains methods which send signal to other forms in order them to be activated. Other classes that depend on the received signal from PDA Gui are in the following. PDA Gui class has these classes. They are:

- **PositionSelectionMenu**: This class is for selecting the way of gathering position of user. These ways are via GPS device, via indicating from map and via manual entry of latitude and longitude.
- **SearchTypeMenu**: This class is for determining search option. The presented options in CoreAccess are “search by category”, “search by activity name” and “browse on map for activity”. Please refer to Analysis report for detailed explanation about choices.
- **SearchByAttributes**: This class is for starting search by activity name and some properties of activity such as date, time, etc.
- **SearchCategory**: This class is for starting search by category which are theatre, cinema, sport and concert.
- **ResultScreen**: This class is for displaying the results in a list. When user selects an item from this list, s/he can see the details of the item or see the item on the map.
- **DetailedInfoScreen**: This class is for illustrating details of activity or place which is selected from result screen.
- **MapScreen**: This class is for displaying the item selected from result screen on the map. In this way user can see the direction and distance of the activity and browse on the map.
- **LanguageManager**: This class is for enabling multilingualism. User can change language in every screen in PDA. PDAGui class controls “change language” signal which comes from this class when it is called.
- **GPSInterpreter**: This class is for getting information from GPS. This class accesses to the GPS device and gets latitude and longitude of the user from GPS device and sends it to the PDA.
- **Map(In PDA part)**: This class is for storing map in PDA. If user runs the program first time, map is loaded to the PDA via this class. If it is not first time, the version is checked. If the original map was changed, it is loaded again. If not, it is remains as it is.
- **MapVersionChecker**: This class is for checking the version of map. As it is mentioned above, It is interacts with Map class. Keep the version of the map and controls the new map version. If the versions are different, it informs that to the PDA.
- **XMLQueryHandler**: This class is for handling the XML files which come to web service and which are sent to the PDA. The gathering information from PDA are added to the XML file and generateXml() function of this class is called. When an XML file come from web server, parseXml() method is called and the file is parsed in that class and the information sent to the PDA.
- **WebServiceCaller**: This class is for calling web service. After XML file generated the web service is called and communication is built.
6.2.2. Request Handler Class Diagram

[Diagram showing the Request Handler Class Diagram with class relationships and methods.]

- `implementation class` XMLQueryParser
  - `USER_QUERY` int
  - `ADMIN_QUERY` int
  - `QUERY` int
  - `createRequestObject` RequestObject
  - `parsXMLRequest` RequestObject
  - `parseAdminRequest` RequestObject
  - `parseRequestObject` RequestObject
  - `formSearchActivityObject` RequestObject
  - `formModifyMapRequest` RequestObject
  - `formModifyNodeRequest` RequestObject
  - `formModifyVehicleRequest` RequestObject
  - `formModifyActivityRequest` RequestObject
  - `formModifyOntologyRequest` RequestObject
  - `generateXMLResponse` RequestObject

- `implementation class` SearchRequest
  - `mapName` string
  - `destination` string
  - `positionLongitude` double
  - `positionLatitude` double
  - `getActivity()`
  - `getTransportation()`
  - `getLocation()`

- `implementation class` MapModificationRequest
  - `mapName` string
  - `modificationType` int
  - `binaryMap` boolean
  - `node`
  - `vehicle`
  - `getModification()`
  - `addNode()`
  - `addVehicle()`
  - `addDataNode()`
  - `deleteNode()`
  - `deleteVehicle()`

- `implementation class` DatabaseManager
  - `dbHost` string
  - `dbPort` string
  - `dbName` string
  - `dbPassword` string
  - `createRequestObject` RequestObject
  - `connectDB()`
  - `makeSQLQuery()`
  - `getResultSet()`
  - `sendResult()`
  - `buildNodeObject()`

- `implementation class` Logger
  - `logMessage` string
  - `logError` string
  - `logException()`

- `implementation class` RequestObject
  - `name` string
  - `type` int
  - `setNodeName()`
  - `getElement()`
  - `setModificationType()`
  - `setResult()`

- `implementation class` WebServiceCommunicationPoint
  - `webServiceCalled()`
  - `start()`
  - `stop()`
  - `directCallTo()`
  - `adminInvocation()`
  - `createInvocation()`
  - `createVirtualObject()`
  - `sendResult()`

- `implementation class`calls

- `UsedBy`

- `UsedBy`

- `UsedBy`
This Part is for generating object according to the request come from user. We have an main class called **RequestObject**. Some classes are inherited from the RequestClass. These objects are in the following:

- **SearchRequest**: This class represents search request of user about. According to the received information, the object initializes itself with the attributes.
- **MapModificationRequest**: This class represents the modification request for map. The database gathers nodes according to instance of this class.
- **IdentificationRequest**: This class represents the authentication request of Administrators and Content Managers. According to the password and id of the staff, it decides whether give permission or not to enter to the system.

When the request object is created, it is firstly sent to **Logger** in order it to be saved in the database for statistical information.

**XMLQueryParser** class is responsible for parsing the received xml messages and forming the request objects using the information in xml.

**DatabaseManager** is the responsible class in dealing with database. According to the RequestObject arrived it forms the required sql queries. Other duties of this class are explained below.

**WebServiceCommunicationPoint** is the class which receives web service calls from the clients. It directs the received xml message to XMLQueryParser class. It returns the result of the request also in xml format.
6.2.3. Activity Search Class Diagram

This part is the class diagram of the server module which finds the activities which the user queries. As mentioned above, `DatabaseManager` deals with database communication. It firstly creates `ActivitySearcher` class. `DatabaseManager` also creates `Activity` and `Place` instances according to the database query result. Each created class inserts itself to `ActivitySearcher` object. Then `ActivitySearcher` finds the appropriate activity and its place according to the request object. Moreover, it uses `OntologyManager` class in order to find related activities in case of not finding any suitable activity. At the end, `Sorter` class gets the results from `ActivitySearcher` and sorts the activities according to a criteria such as alphabetic order of activity name.
6.2.4. Transportation Class Diagram

This part is the class diagram of the server module which finds the transportation options and indicates the paths of these vehicles. Again DatabaseManager has an active role. It creates
Graph, **NodeObject** and **NodeLink** object. Each created NodeObject and NodeLink inserts itself into created Graph object. Then Graph object sends its nodes to **ShortestPathFinder** in order to find the shortest path. Also the nodes are sent to **Transportation** class. Transportation class adds some vehicle objects if necessary. Then these nodes are merged in **Map** class to form the complete map.

### 6.2.5. Web Service Class Diagram

![WebServiceClassDiagram]

**WebServiceCommunicationPoint** class received calls from **AdminInterface** and **ContentManagerInterface**. These interface classes generates the queries according to written information gathered from the staff.

### 6.2.6. Database Class Diagram

![DatabaseClassDiagram]

For staff queries, **DatabaseManager** passes the request objects to **ContentManagerDatabaseModule** and **AdminDatabaseModule** objects. These objects forms the staff specific queries and manages the returned result sets in a way that it can be send as xml message.
6.3. Sequence Diagrams

6.3.1. Search by Attributes

In order to perform a search according to the attributes of the wanted place or activity, PDA user accesses to the PDAGui. When PDAGui is started, it initializes itself. In this initialization all of the user interface components are created and their content are organized. After completing the initialization, PDAGui is ready to go to PositionSelectionMenu. To show and make the necessary preparation, initializeGui method of PositionSelectionMenu is called. Now user interface is visible and shows the position selection alternatives to the user. For the details of position selection please see “4.3.4. Retrieve Position” part. When the user selects an option and clicks the next button, an ActionListener in PositionSelectionMenu catches this action and invokes actionPerformed method. In the actionPerformed method the position information is retrieved and it is sent to PDAGui which is the responsible class for coordinating the user interfaces. Now the system has the position information of the user.
The next step is the selection of search type. For this purpose, *initializeGui* method of *SearchTypeMenu* is called. Upon receiving this call, *SearchTypeMenu* makes necessary initializations and shows the interface to the user. User selects an option; in this case this option is Search-By-Attributes. This select event is caught. Since the search type is Search-By-Attributes, *toSearchByName* method of the *PDAGui* is called. This call triggers *initializeGui* method call for the *SearchByAttributes* object. In this method, the arrangements of the contents of SearchByAttributes menu are completed and the menu is shown to the user in order to get the attributes of the wanted destination or place. When the user finishes entering the attribute information, he or she clicks on the search button which invokes the *actionPerformed* method of *SearchByAttributes* class. The gathered information is organized and sent to the PDAGui class via *sendDataToXMLQueryHandler* method. *PDAGui* passes this information to *XMLQueryHandler* by invoking its *generateXml* method. *XMLQueryHandler* rearranges the search information in XML format. This new format of search query is sent to *WebServiceCaller* by invoking *callService* method. *WebServiceCaller* initiates a call to Server and gets its result. For the details of web service communication please see the part “4.3.7. Web Service Communication”.

The returned search result is in XML format and should be converted to human understandable format. Therefore, *parseXml* method of the *XMLQueryHandler* is called. As *XMLQueryHandler* parses the messages, it invokes the *addResult* method of *ResultScreen*. This method is called multiple times because for each item of the search result *addResult* is called. When all of the results are sent to *ResultScreen*, *toResultScreen* method of *PDAGui* is called. In fact, we could directly call the *displayResult* method of *ResultScreen* instead of indirectly calling *toResultScreen*. However, we have chosen the second one for the sake of consistency of user interface management. *PDAGui* calls the *displayResult* method of *ResultScreen* and *ResultScreen* shows the result as a list.

In order to perform another query, user clicks on “go to main menu” button. This action triggers the call of *toSearchTypeMenu*. When *PDAGui* receives this call, it will invoke *initializeGui* method of *SearchTypeMenu* object. An important point here is that we skipped choosing the position selection method in the second and next searches in order not to bother the user with the unnecessary repetition of choosing position selection method. However, if the user wants, he or she can change the position selection method from the settings menu.
6.3.2. Search by Category

Retrieving position information is similar to the one on Search By Attributes. We included it in the diagram for the completeness of the search.

Now we have the position information (If the position selection method is “via GPS”, then the position information is continuously updated).

Again similar to the previous section, initializeGui method of SearchTypeMenu is called in order SearchTypeMenu to prepare itself and show the search type selection menu. In this sequence user selects Search-By-Category. SearchTypeMenu obtains this choice and calls toSearchByCategory method of PDAGui which calls initializeGui method of SearchCategory class. SearchCategory class prepares a menu on which a number of categories are shown to the user. The user selects one of the options (cinema, theatre, concert hall, sports center). This information is retrieved by actionPerformed method of the SearchCategory class and is sent to PDAGui via sendDataToXMLQueryHandler method. PDAGui forwards the category information to XMLQueryHandler by invoking generateXML method. We preferred sending
the query indirectly through PDAGui because we wanted to preserve the coordinator role of PDAGui so that information flow between user interface and internal part of our system is easily maintainable. XMLQueryHandler puts the category information into XML format and passes the data to WebServiceCaller by invoking callService method. WebServiceCaller calls the web service method of server and returns the result by calling parseXML method of XMLQueryHandler in order to parse it and retrieve the necessary information.

The steps for forming the result screen are the same as the steps in Search-By-Attributes. We included it for the completeness of the search.

6.3.3. Search on Map

The explanation of retrieving position selection method is skipped. Please refer to “4.3.1. Search by Attributes” for the explanation.

In this sequence diagram, after the SearchTypeMenu is initialized, the user selects Search-On-Map option. SearchTypeMenu informs PDAGui about this request by calling toMapScreen. At this moment user browses the map by applying zoom-in/zoom-out or scroll down/up/right/left. When the user finds the place he is looking for, clicks on the place which will initiate the
search. The coordinates of the place is sent to XMLQueryHandler with generateXML method. XMLQueryHandler converts the coordinate information in the XML format defined by the XSD given in attachment. Then it sends this XML message to WebServiceCaller with callServiceMethod. Another duty of XMLQueryHandler is to parse the result of the search query received by WebServiceCaller. As the results are parsed, they are added to ResultScreen. When the result processing is finished, toResultScreen method of PDAGui is called to switch to result menu.

The user can select one of the results from the result list to see it on map. In this case, ResultScreen invokes toMapScreen method of PDAGui. Then, PDAGui will initialize MapScreen for displaying the result on map by initializeGui method. Upon receiving this call, MapScreen will arrange the viewed part of the map and display it.

6.3.4. Retrieve Position

In this part, how position information is gathered is explained. There are three ways of gathering position: from GPS, manual and from map.

6.3.4.1. Via GPS Receiver

This process is realized automatically transparent to the user when the user makes a search. PositionSelectionMenu is informed about the position request. Then getLatitude and getLongitude methods are called respectively.
6.3.4.2. Via Manually Entering

This process is mainly used for testing purposes. The user enters the longitude and latitude values. These values are assigned to the properties of PositionSelectionMenu for retrieval of position information later.

6.3.4.3. Via Browsing Map
When the GPS device is not available, user can select the position on the map. We will also add a functionality providing users with a text based search option which enables to benefit from well-known places while searching his/her position.

In order to show map to the user, PositionSelectionMenu invokes toMapScreen method of PDAGui. Then, PDAGui calls the display method of MapScreen. After browsing the map, the user selects a place. The position is obtained by getXCoordinate and getYCoordinate methods of MapScreen. According to these values, by calling calculateLatitudeAndLongitude longitudes and latitudes are calculated and sent back to PositionSelectionMenu.

6.3.5. See Detailed Information

This sequence diagram describes which calls are performed when the user wants to see the details of results in the result list. The details includes the name, contact information, address and comments about the place or activity.

On the ResultScreen user selects an item and clicks on detailed info button. This invokes buttonPressed method of ResultScreen. ResultScreen sends the information of selected item to PDAGui with toDetailedInfoScreen. PDAGui forwards this information to DetailedInfoScreen. When DetailedInfoScreen receives displayDetail call, it performs necessary initializations according to selected result information and displays them to the user. When the user clicks on back button in order to see the details of other results or in order to see a result on map, DetailedInfoScreen calls toResultScreen method of PDAGui. Then PDAGui will make visible ResultScreen again.
6.3.6. See Transportation on Map

From the ResultScreen user may want to see the transportation information to a place in the result list. This transportation information defines the possible paths to the destination by also indicating the vehicle types available along with their time and cost estimations.

By selecting and clicking on SeeOnMap button, the user initiates the transportation display process. ResultScreen calls requestMap method of PDAGui with the information of selected item as parameters. PDAGui forwards this information to MapScreen and tells it to display the corresponding map. We will use caching of maps in order to increase the speed. Therefore, we should consider whether the map available in PDA should be updated or not. This task is independent of MapScreen. MapScreen just request the map from PDAMap with getCurrentDisplay method which consults MapVersionChecker for checking the version of the map. If the version is not sufficient (the map needs update), MapVersionChecker calls generateXML method of XMLQueryHandler. WebServiceCaller calls the server and returns the map. The version of the current map updated and displayed to the user.

When the user clicks on back button, PDAGui is informed with toResultScreen method and calls displayResult method of ResultScreen. Then, the results available at the beginning will be shown. Now the user can select another result from the result list for obtaining further details.
6.3.7. Web Service Communication
We have built our architecture over web services in order to avoid platform and language
dependency. Thanks to web services, we are also planning to make our software available to
everyone. In this way, with any program supporting web services an application which uses our
server side can be developed. In this section we describe how our client applications access to
the server.

6.3.7.1. PDA Web Service
WebServiceCaller calls webService Called method of WebServiceCommunicationPoint with
which resides on server side. The request information which is in XML format is passed as
parameter. WebServiceCommunicationPoint delivers this request to related modules. When the
result arrives, the result is sent back to WebServiceCaller.

6.3.7.2. Administrator Web Service
AdminInterface calls webService Called method of WebServiceCommunicationPoint with
which resides on server side. The request details which is in XML format is passed as
parameter. WebServiceCommunicationPoint delivers this request to related modules. When the
result arrives, the result is sent back to WebServiceCaller.
6.3.7.3. Content Manager Web Service

`webServiceCalled` method of `WebServiceCommunicationPoint` which is on server side is called by `ContentManagerInterface` invokes. The request details which is in XML format is passed as parameter. `WebServiceCommunicationPoint` delivers this request to related modules. When the result arrives, the result is sent back to `WebServiceCaller`.

6.3.8. Handle XML

Since `WebServiceCommunicationPoint` receives the request in xml format and sends the result again in xml format, xml data should be processed. This is achieved by `XmlQueryParser`. `XmlQueryParser` parses the xml data by `parseXmlRequest` and sends the result of request to `WebServiceCommunicationPoint` by `sendResult` method.
6.3.9. Find Vehicles

In this part, the sequence in server side for finding the vehicles. When XMLQueryParser gets the request in xml format, it creates RequestObject to be used by other modules. One of these modules is logger module. RequestObject calls saveLogsToFile method of Logger and Logger extracts the necessary information and saves the request. The other module which uses RequestObject is database module. RequestObject first connects to the database via connectDB method. Then it invokes makeSqlQuery method of DatabaseManager so that necessary sql queries are created according to the request information. DatabaseManager gets the required information from database and creates Graph, NodeObject and NodeLink objects. Inserting the NodeObject and NodeLink objects into Graph is also responsibility of DatabaseManager. In order to indicate that insertion process is finished, it lastly calls insertionCompleted method of Graph. Then Graph class constructs a vector map from these links, and also sends the node related objects to ShortestPathFinder in order it to find the shortest path via the method findShortestPath. Next, Graph notifies Transportation for finding the appropriate transportation vehicles. Both Transportation and ShortestPathFinder passes the path and transportation knowledge to Map object. Lastly, MapManipulator highlights the path to the destination and calls generateXML method of XMLQueryParser which converts the constituted objects to XML format.
6.3.10. Find Results

Constructing the request objects is the same as in “4.3.9 Find Vehicle”. They are included for the completeness of the process only.

This time DatabaseManager builds different objects. These are ActivitySearcher, Activity and Place objects. As Place and Activity objects are created they are inserted into ActivitySearcher by insertPlace and insertActivity methods respectively. When the creation of the objects is finished, DatabaseManager calls insertionCompleted method. If the exact match is not found, ActivitySearcher requests related items by first loading the ontology by loadOntology method and next calling findRelatedItem method of OntologyManager. According to the new relations activity search is performed again. Then the results are sent to Sorter class in order it to sort them according to different criteria such as cost, time, distance etc. Lastly, Sorter class calls generateXmlResponse method of XMLQueryParser class so that the response objects are translated into xml form to be send thorough web service.
6.3.11. Content Manager Database

Similar to the other processes XMLQueryParser generates RequestObject. RequestObject initializes itself by start method. RequestObject next calls connectDB and makeSqlQuery method of DatabaseManager. Then DatabaseManager extracts the information from database and sends them to CMDatabaseManager. After CMDatabaseManager completes its job, it calls generateXmlResponse method of XmlQueryParser which performs the conversion between response objects and xml.

6.3.12. Administrator Database
After `RequestObject` is created by `start` method, `RequestObject` initializes itself. Next it sends signal to `DatabaseManager` to connect to database and the parsed data is passed to `DatabaseManager` by `makeSqlQuery`. `DatabaseManager` sends the extracted data to `CMDatabaseManager` for further process of the data. When this process is finished, `generateXmlResponse` method is called and `XmlQueryParser` converts the response objects into xml format.
6.4. Activity Diagrams

6.4.1. User Activity Diagram

In this activity diagram, User Interface Module interacts with Webservice Module. PDA user activates the system with an external force for instance run the application in PDA. Then, user selects his/her position with one of three selection options which are selection position from map while zooming in/out and scaling, from GPS device and entering the latitude and
longitude manually. After setting position information properly, user passes to the following screen which is searching. In this phase, user has 3 selection options which are searching by attributes, searching by category and searching on map. In the first two options, user has to fill some fields. For instance, if user wants to search an activity according to date, he/she only fills date field in search by attributes and do the search. However, it is a bit different in searching on map. In this option, map is directly displayed and user can navigate on that map by zooming in/out, scaling, etc. When user indicates a point on map and searches for activity places, the information of the point is collected in an XML file. At this point, WebService Module occurs. WebService is called and gathered XML data is handled by an XML parser. Parsed objects sent to the database and manipulated. After relevant processes are done in database manager side, the results return to the PDA user with the help of WebService. List of items are illustrated on the PDA screen. In that case, user has two possible options. One of them is picking one item from the list and seeing the detailed information of selected activity place or activity. The other one is viewing transportation options for an activity place. From both interfaces, user can return to the result screen. For a new search, user has to jump to the selection menu from result list.

6.4.2. Web Service Activity Diagram
The above diagram demonstrates the process on activity and GIS data in server part. This diagram contains has 3 modules interacted with themselves which are WebService module, user interface module and database module. The process sequence starts by calling the webservice. The retrieved information in XML format parsed and relevant data structures are generated in managing database side. The next step is generating sql queries in order to be sent to database. In this phase, there are 3 different search queries according to the contents. They are activity information result returned query, map data result returned query and queries about history of searches. After formation of the search queries, they are sent to the database in order to be executed. At the end the result list is packed as an XML file and displayed in the PDA screen of user.

6.4.3. Administrator and Content Manager Activity Diagram

The following diagram illustrates the admin and content manager. Admin/Content Manager Interface module and Webservice Module have an interaction between each other. In order to enter the system, admin and content manager login to the system. After login, a screen for changing the map, activity place, activity appears. They can fill the text fields on the screen and send it to the webservice. Since we use webservice, we sent the data in the XML format. After parsing the received information, database manipulates these data and returns results for instance activity insertion success. Like the above diagrams this diagram packed the result in an XML format and sends it to the user interface. Admin and Content Manager can make different modifications in database. Finally they can exit from the system by logging out.
6.5. Collaboration Diagrams

Our application CoreAccess supply the user so many options in finding the appropriate activity place and activity itself. Therefore, different collaboration diagrams are generated in order to illustrate the actions and event of the user. Moreover, the system has some signals that wake up a module. Thus, system functions in an order. In addition to these, system administrator and content manager can activate some modules. In the following diagrams, the processes and operations are explained clearly.
6.5.1. Search by Attributes

This collaboration diagram illustrates one of the user activities which is searching an activity with one of the attributes which are the name, place, date, time etc. Before searching, user has to determine its position via GPS device, manually or via map. After that, user can search type which are search by attributes, search by category and search activity on map and activate toSearchByAttributes() method in PDAGui class. Then, detailed search begins. After typing the attributes, data is sent to the XMLQuery handler class. In there XML file is generated with relevant information. Subsequent to this, WebService is called. Server is responsible for the tasks between the time sent XML and the time returned results as XML. Returned results are parsed in XMLQueryHandler class and listed results are seen on the result screen. With the help of the user external force, toDetailedInfo() method is activated and the interface switch to the DetailedInfo. User can returned to the result list again and can see the transportation options and the place of activity on map by calling requestMap() method in PDAGui. The MapVersionChecker is activated before viewing the map on the pda screen. If there is not coherence between current map version in the pda and last updated one, new map is requested from server by calling callService method. At the end of the processes done on server, new
map is gathered and illustrated on the pda screen. Moreover, for multilingualism user has option for selecting language.

6.5.2. Search by Category

This diagram illustrates one of the types for searching activity which is based on the category which are cinema, theatre, concert hall, sport center and exhibition hall. Data flow is like the above. One important difference is that user does not know about a specific activity. Instead of SearchByAttributes class, SearchCategory class is put in the diagram.
6.5.3. Search on Map

This diagram demonstrates the last type of search which is search on map. It has same data flow. However, the accessing order of map is different. When user selects a type in SearchTypeMenu class, MapScreen class is activated. User can navigate and make a search on the map by zooming in/out, scaling. When indicatorPressed() method is activated by an external force, it sends information to the XMLQueryHandler. XML file is generated and sent to the server. WebService is called. The following step is displaying the result list on the screen. Therefore addResult() method in ResultScreen class and toResultScreen() method in PDAGui is activated. After listing, seeing detailed info is same as above.
6.5.4. Transportation Manipulation on Server Part

This part of application is about the transportation manipulation in webservice side. When webservice is called by the pda user, WebServiceCaller class initiates the processes in server part. Received XML file is parsed in XMLQueryParser and RequestObject which is an activity is created. After that, search queries of coming objects are generated and sent to the database. While database searches for query results, it generates nodeobjects and nodelinks in order to find shortest path and distance between activity place and user. At the end of this process, graph of the paths is obtained. With the help of the extracted graph, shortest path and transportation options are found. Found paths are matched in Map class and MapManipulator highlight the indicated paths. At the end of map manipulation, this map data is packed in XMLQueryParser as XML file and sent to the user with WebServiceCommunicationPoint. The result list is received by the pda user. Finally, CoreAccess holds some statistics about search items for feedback.
6.5.5. Activity and Place on Server Part

The above diagram represents the requested activity manipulation in server side. The coming XML file from Pda user to WebServiceCommunicationPoint class sent to the XMLQueryParser. The file is parsed and a new RequestObject is created. New query for searching activity is generated with this object. The relevant activities with their places are found and listed. Moreover, if result list is empty, CoreAccess consults to the ontology reasoner in order to serve alternative choices to user. After all searches are done the activities are sorted according to the some attributes such as path distance, alphabetic order, etc. Moreover our system keeps some logs about the searched activities and items for feedback.
6.5.6. Administrator and Content Manager Interface Communication

In this collaboration diagram, the interaction between content manager and admin is demonstrated. Content manager and admin communicate with the system via the ContentManagerInterface and AdminInterface classes. The entered data to the interfaces are sent to the server as an XML file. In this phase, XMLQueryParser class parses the file and generates a request object according to the received information. After the queries are formed according to the generated object, queries are sent to the database and manipulated. At the end, the results are obtained and form a return XML file in XMLQueryParser. The file is sent to the user interfaces of the content manager and admin via webservice.

7. SYNTAX DEFINITION

7.1. XML File Representation

In this part, we have defined the structure of the XML messages that will be used in providing the communication between server and the other clients. Our design lies on web service technology so we used XML in order to make processes communicate with each other. The message syntax is composed of User Requests, Administrator Requests, Content Manager Requests, and Responses for Users, Administrators and Content Managers. Details of the message syntax can be found below:
<?xml version="1.0" standalone="yes"?>
<MessagingSyntax>

<!-- USER REQUESTS -->

<!-- USER REQUESTS -->

<!-- USER REQUESTS -->

<!-- ADMIN REQUESTS -->

<!-- ADMIN REQUESTS -->

<!-- ADMIN REQUESTS -->

</ViewPlaceListRequest>

<ViewPlaceListRequest session=""/>

<ModifyPlaceRequest name="" mapName="" session=""/>

<UploadPlace>mapData</UploadPlace>

<DeletePlace/>

<ModifyPlace newName=""/>

</ModifyPlaceRequest>

<ViewPlaceListRequest session=""/>

<ModifyPlaceRequest name="" mapName="" session=""/>

<UploadPlace>mapData</UploadPlace>

<DeletePlace/>

<ModifyPlace newName=""/>

</ModifyPlaceRequest>

</MessagingSyntax>
<ViewVehicleListRequest session=""/>

<ModifyVehicleRequest name="" mapName="" session=""/>
  <AddVehicle>vehicleData</AddVehicle>
  <DeleteVehicle/>
  <RenameVehicle newName=""/>
</ModifyVehicleRequest>

<ViewOntologyListRequest session=""/>

<ModifyOntologyRequest name="" mapName="" session=""/>
  <AddOntology>ontologyData</AddOntology>
  <DeleteOntology/>
  <UpdateOntology>ontologyData</UpdateOntology>
</ModifyOntologyRequest>

<ViewCMListRequest session=""/>

<ViewCMStatisticsRequest session=""/>

<ModifyCMRequest name="" session=""/>
  <AddCM> CMDATA AS XML </AddCM>
  <DeleteCM/>
  <UpdateCM> CMDATA AS XML </UpdateCM>
</ModifyCMRequest>

<MonitorRequest>
  <UserLog/>
  <ActivityLog/>
  <ActivityPlaceLog/>
  <CategoryLog/>
</MonitorRequest>

!-- CONTENT MANAGER REQUESTS -->

<LoginRequest type="CM" name="" passwd=""/>

<ModifyActivityRequest name="" mapName="" session=""/>
  <AddActivity>nodeData</AddActivity>
  <DeleteActivity/>
  <RenameActivity newName=""/>
  <ModifyActivityDescription>
    <DeleteDescription descrName=""/></DeleteDescription>
    <UpdateDescription newDescp=""/></UpdateDescription>
    <AddDescription newDescp=""/></AddDescription>
  </ModifyActivityDescription>
</ModifyActivityRequest>

<ModifyActivityPlaceRequest name="" mapName="" session=""/>
  <UpdatePhoneNumber phone=""/></UpdatePhoneNumber>
  <UpdateAddress address=""/></UpdateAddress>
  <UpdateEmail email=""/></UpdateEmail>
</ModifyActivityPlaceRequest>
<!-- RESPONSE FOR USERS -->

<SearchLocationResponse>
  <Place name="" type="" phone="" email="" currentPosition="">
    <Address>address</Address>
    <Path color="">
      nodeListSeperatedByCommas
    </Path>
    <DrawPoint latitude="" color=""></DrawPoint>
    <Distance type="" quantity=""></Distance>
  </Place>
  ....
</SearchLocationResponse>

<SearchCategoryResponse name="" currentPosition="">
  <CategoryInfo name="">
    <Place name="" type="" phone="" email="" position="">
      <Address>address</Address>
      <Path color="">
        nodeListSeperatedByCommas
      </Path>
      <DrawPoint longitude="" latitude="" color=""></DrawPoint>
      <Distance type="" quantity=""></Distance>
    </Place>
    <Activity name="" type="" time="" date="">
      <descr>descr</descr>
    </Activity>
  ....
</CategoryInfo>
</SearchCategoryResponse>

<TransportationAlternativesResponse fromPos="" toPos="">
  <VehicleList>
    <Vehicle name="" time="" type="" fromPos="" toPos="" min="" cost="">
      <Path color="">
        nodeListSeperatedByCommas
      </Path>
      <Distance type="" quantity=""></Distance>
    </Vehicle>
  </VehicleList>
  ....
</TransportationAlternativesResponse>

<MapResponse name="" zoomLevel="" position="">
  <MapData>mapdata</MapData>
</MapResponse>

<!-- RESPONSE FOR ADMIN -->

<LoginResponse type="CM | A" isSuccessful=""
  session="">message</LoginResponse>

<ViewActivityListResponse isSuccessful="">

<Activity name="" location="" type="" address="" date="" time="" phone=""">description</Activity>

...</ViewActivityListResponse>

<ModifyActivityResponse name="" isSuccessful="">
  message
</ModifyActivityResponse>

<ViewMapListResponse isSuccessful="">
  <MapData name="" zoomLevel="" mapdata</MapData>
</ViewMapListResponse>

<ModifyMapResponse name="" isSuccessful="">
  message
</ModifyMapResponse>

<ViewPlaceListResponse isSuccessful="">
  <Place name="" type="" phone="" email="" destinationPosition="">
    <Address>address</Address>
    <Path color="">
      nodeListSeperatedByCommas
    </Path>
    <DrawPoint longitude="" latitude="" color=""></DrawPoint>
    <Distance type="" quantity=""></Distance>
  </Place>
  ...
</ViewPlaceListResponse>

<ModifyPlaceResponse isSuccessful="">
  message
</ModifyPlaceResponse>

<ViewVehicleListResponse isSuccessful="">
  <Vehicle name="" time="" type="" fromPos="" toPos="" min="" cost="">
    <Path color="">
      nodeListSeperatedByCommas
    </Path>
    <Distance type="" quantity=""></Distance>
  </Vehicle>
  ...
</ViewVehicleListResponse>

<ModifyVehicleResponse isSuccessful="">
  message
</ModifyVehicleResponse>

<ViewOntologyListResponse isSuccessful="">
  <Ontology name="" location="">ontology</Ontology>
  ...
</ViewOntologyListResponse>

<ModifyOntologyResponse isSuccessful="">
  message
</ModifyOntologyResponse>
<ViewCMListResponse isSuccessful="">
<CM name="" passwd="" email="" phone="" address="" id=""
company="" regDay="/">
....
</ViewCMListResponse>

<ViewCMStatisticsResponse isSuccessful="">
<Action name="" query="" cmID="/"></Action>
...
</ViewCMStatisticsResponse>

<ModifyCMResponse isSuccessful="">
message
</ModifyCMResponse>

<MonitorResponse isSuccessful="">
<UserLog>log</UserLog>
...
<ActivityLog>log</ActivityLog>
...
<ActivityPlaceLog>log</ActivityPlaceLog>
...
<CategoryLog>log</CategoryLog>
...
</MonitorResponse>

<!-- RESPONSE FOR CONTENT MANAGERS -->
<LoginResponse type="CM | A" isSuccessful=""
session="/">message</LoginResponse>

<ViewActivityListResponse isSuccessful="">
<Activity name="" location="" type="" address="" date="" time=""
phone="/">description</Activity>
...
</ViewActivityListResponse>

<ModifyActivityResponse isSuccessful="">
message
</ModifyActivityResponse>

<ModifyActivityPlaceResponse isSuccessful="">
message
</ModifyActivityPlaceResponse>

</MessagingSyntax>

7.2. XSD of the System

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<xs:import namespace="http://www.w3.org/2001/XMLSchema-instance"
schemaLocation="xsi.xsd"/>
<xs:element name="SearchLocationRequest">
  <xs:complexType>
    <xs:attribute name="currentPosition" use="required"/>
    <xs:attribute name="destinationPosition" use="required"/>
    <xs:attribute name="map" use="required"/>
    <xs:attribute name="name" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="SearchCategoryRequest">
  <xs:complexType>
    <xs:attribute name="currentPosition" use="required"/>
    <xs:attribute name="map" use="required"/>
    <xs:attribute name="name" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="TransportationAlternativesRequest">
  <xs:complexType>
    <xs:attribute name="fromPos" use="required"/>
    <xs:attribute name="map" use="required"/>
    <xs:attribute name="toPos" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="MapRequest">
  <xs:complexType>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="position" use="required"/>
    <xs:attribute name="zoomLevel" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="DistanceCalculationRequest">
  <xs:complexType>
    <xs:attribute name="loc1" use="required"/>
    <xs:attribute name="loc2" use="required"/>
    <xs:attribute name="map" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="LoginRequest">
  <xs:complexType>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="passwd" use="required"/>
    <xs:attribute name="type" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="ModifyActivityRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="AddActivity"/>
      <xs:element ref="DeleteActivity"/>
      <xs:element ref="RenameActivity"/>
      <xs:element ref="ModifyActivityDescription"/>
    </xs:sequence>
    <xs:attribute name="mapName" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="session" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="AddActivity" type="xs:NCName"/>
<xs:element name="DeleteActivity"/>
<xs:element name="ModifyNode">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="AddNode"/>
      <xs:element ref="DeleteNode"/>
      <xs:element ref="RenameNode"/>
    </xs:sequence>
    <xs:attribute name="name" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="AddNode" type="xs:NCName"/>
<xs:element name="DeleteNode"/>
<xs:element name="RenameNode">
  <xs:complexType>
    <xs:attribute name="newName" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="ModifyOntologyRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="AddOntology"/>
      <xs:element ref="DeleteOntology"/>
      <xs:element ref="UpdateOntology"/>
    </xs:sequence>
    <xs:attribute name="mapName" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="session" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="AddOntology" type="xs:NCName"/>
<xs:element name="DeleteOntology"/>
<xs:element name="UpdateOntology" type="xs:NCName"/>

<xs:element name="ModifyPlaceRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="UploadPlace"/>
      <xs:element ref="DeletePlace"/>
      <xs:element ref="ModifyPlace"/>
    </xs:sequence>
    <xs:attribute name="mapName" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="session" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="UploadPlace" type="xs:NCName"/>
<xs:element name="DeletePlace"/>
<xs:element name="ModifyPlace">
  <xs:complexType>
    <xs:attribute name="newName" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="ModifyVehicleRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="AddVehicle"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element ref="DeleteVehicle"/>
<xs:element ref="RenameVehicle"/>
</xs:sequence>
<xs:attribute name="mapName" use="required"/>
<xs:attribute name="name" use="required"/>
<xs:attribute name="session" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="AddVehicle" type="xs:NCName"/>
<xs:complexType/>
</xs:element>
<xs:element name="DeleteVehicle">
<xs:complexType>
</xs:element>
<xs:element name="RenameVehicle">
<xs:complexType>
<xs:attribute name="newName" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="MonitorRequest">
<xs:complexType>
<xs:sequence>
<xs:element ref="UserLog"/>
<xs:element ref="ActivityLog"/>
<xs:element ref="ActivityPlaceLog"/>
<xs:element ref="CategoryLog"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ViewActivityListRequest">
<xs:complexType>
<xs:attribute name="session" use="required"/> 
</xs:complexType>
</xs:element>
<xs:element name="ViewCMListRequest">
<xs:complexType>
<xs:attribute name="session" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="ViewCMStatisticsRequest">
<xs:complexType>
<xs:attribute name="session" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="ViewMapListRequest">
<xs:complexType>
<xs:attribute name="session" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="ViewOntologyListRequest">
<xs:complexType>
<xs:attribute name="session" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="ViewPlaceListRequest">
<xs:complexType>
<xs:attribute name="session" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="ViewVehicleListRequest">
<xs:complexType>
<xs:attribute name="session" use="required"/>
</xs:complexType>
</xs:element>
</xs:element>
<xs:element name="ModifyActivityPlaceRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="UpdatePhoneNumber"/>
      <xs:element ref="UpdateAddress"/>
      <xs:element ref="UpdateEmail"/>
    </xs:sequence>
    <xs:attribute name="mapName" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="session" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="UpdatePhoneNumber">
  <xs:complexType>
    <xs:attribute name="phone" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="UpdateAddress">
  <xs:complexType>
    <xs:attribute name="address" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="UpdateEmail">
  <xs:complexType>
    <xs:attribute name="email" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="SearchLocationResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Place"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="SearchCategoryResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="CategoryInfo"/>
    </xs:sequence>
    <xs:attribute name="currentPosition" use="required"/>
    <xs:attribute name="name" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="CategoryInfo">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="Activity"/>
      <xs:element ref="Place"/>
    </xs:choice>
    <xs:attribute name="name" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="TransportationAlternativesResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="VehicleList"/>
    </xs:sequence>
    <xs:attribute name="fromPos" use="required"/>
    <xs:attribute name="toPos" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="VehicleList">
    <xs:complexType mixed="true">
        <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="Vehicle"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="MapResponse">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="MapData">
                <xs:attribute name="name" use="required"/>
                <xs:attribute name="position" use="required"/>
                <xs:attribute name="zoomLevel" use="required"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<xs:element name="LoginResponse">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:NCName">
                <xs:attribute name="isSuccessful" use="required"/>
                <xs:attribute name="session" use="required"/>
                <xs:attribute name="type" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="ModifyActivityResponse">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:NCName">
                <xs:attribute name="isSuccessful" use="required"/>
                <xs:attribute name="name"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="ModifyCMResponse">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:NCName">
                <xs:attribute name="isSuccessful" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="ModifyMapResponse">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:NCName">
                <xs:attribute name="isSuccessful" use="required"/>
                <xs:attribute name="name" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="ModifyOntologyResponse">
    <xs:complexType>
<xs:simpleContent>
  <xs:extension base="xs:NCName">
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="ModifyPlaceResponse">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:NCName">
        <xs:attribute name="isSuccessful" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="ModifyVehicleResponse">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:NCName">
        <xs:attribute name="isSuccessful" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="MonitorResponse">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="ActivityLog"/>
      <xs:element ref="ActivityPlaceLog"/>
      <xs:element ref="CategoryLog"/>
      <xs:element ref="UserLog"/>
    </xs:choice>
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="ViewActivityListResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Activity"/>
    </xs:sequence>
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="ViewCMListResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="CM"/>
    </xs:sequence>
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="CM">
  <xs:complexType>
    <xs:attribute name="address" use="required"/>
    <xs:attribute name="company" use="required"/>
    <xs:attribute name="email" use="required"/>
    <xs:attribute name="id" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="passwd" use="required"/>
    <xs:attribute name="phone" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="ViewCMStatisticsResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Action"/>
    </xs:sequence>
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="Action">
  <xs:complexType>
    <xs:attribute name="cmID" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="query" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="ViewMapListResponse">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="MapData">
        <xs:attribute name="isSuccessful" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="ViewOntologyListResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Ontology"/>
    </xs:sequence>
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="Ontology">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:NCName">
        <xs:attribute name="location" use="required"/>
        <xs:attribute name="name" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="ViewPlaceListResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Place"/>
    </xs:sequence>
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="ViewVehicleListResponse">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Vehicle"/>
    </xs:sequence>
    <xs:attribute name="isSuccessful" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="ModifyActivityPlaceResponse">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:NCName">
        <xs:attribute name="isSuccessful" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="UserLog" type="xs:string"/>
<xs:element name="ActivityLog" type="xs:string"/>
<xs:element name="ActivityPlaceLog" type="xs:string"/>
<xs:element name="CategoryLog" type="xs:string"/>
<xs:element name="Place">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Address"/>
      <xs:element ref="Path"/>
      <xs:element ref="DrawPoint"/>
      <xs:element ref="Distance"/>
    </xs:sequence>
    <xs:attribute name="currentPosition"/>
    <xs:attribute name="destinationPosition"/>
    <xs:attribute name="email" use="required"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="phone" use="required"/>
    <xs:attribute name="position"/>
    <xs:attribute name="type" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="Address" type="xs:NCName"/>
<xs:element name="DrawPoint">
  <xs:complexType>
    <xs:attribute name="color" use="required"/>
    <xs:attribute name="langitude" use="required"/>
    <xs:attribute name="latitude" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="Activity">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="descr"/>
    </xs:sequence>
    <xs:attribute name="address"/>
    <xs:attribute name="date" use="required"/>
    <xs:attribute name="location"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="phone"/>
    <xs:attribute name="time" use="required"/>
    <xs:attribute name="type" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="descr" type="xs:NCName"/>
<xs:element name="Vehicle">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Path"/>
      <xs:element ref="Distance"/>
    </xs:sequence>
    <xs:attribute name="cost" use="required"/>
    <xs:attribute name="fromPos" use="required"/>
  </xs:complexType>
</xs:element>
8. CONCLUSION

Preparing the complete design report of CoreAccess was a very profitable milestone for CoreTech. Designing the project with several diagrams in the initial design helped us with making our minds more clear about project. Then in the complete design, we have prepared the graphical user interfaces of our project. Moreover, we have explained the diagrams clearly. As a result, we believe we did good work by forming the complete design report and we will have the benefit of it soon.

9. APPENDIX

9.1. Updated Gantt Chart

Updated Gantt Chart can be found at the next page.