ClimbPlanner

Final Design Report

By

Quattro Group

Authors:

Gökçen Güner

Mehmet Aktaş

Mehmet Uygar Akgül

Assistant:

Ali Orkan Bayer

19.01.2009

Contents:

1. Introduction

For the last 20 years Geographic Information Systems have had a big role in the daily life. In contemporary life, GIS systems are used in several aspects such as scientific investigations, resource management, asset management, archaeology, environmental impact assessment, urban planning, cartography, criminology, geographic history, marketing, logistics etc. more and more. . This shows the growth of the importance of GIS technologies in everyday life. It will have a more widespread usage in real life in the future because it is a technology that concerns lots of important aspects of life. However, there has been a lack in software solutions for assistance for climbers in finding their safe routes during climbing their target mountain. Since climbing a mountain requires a great amount of careful planning, the assistant software should be very accurate and should consider the requirements of the climbers. This project is aiming to develop climbing assistant software that meets the mentioned criteria in order to make up for the deficiency in the market.

1.1 Problem Definition

In mountain climbing experience is a very important point for a climber, because mountain conditions are formidable most of the time. However, not only experience but also equipment and physical condition of climber have a vital importance for a safe climb. Moreover, since climbing is a group activity, deficiencies of any member of the group will affect the whole activity program.

We need to define a good solution for climbers considering all these parameters together with geographic and meteorological parameters. Our main problem is defining a desirable route for the climbing group after getting required information from users. However, the user information is not enough to define the path. We will also need to obtain geographical and weather conditions somehow. There is no solution for obtaining geographical data and weather data together. We need to get them separately and then combine them to calculate the accurate route. The software we are designing will offer some options for the user to choose how the program will get this information.

Another problem to solve is visualization and simulation of the route and climbing area. The outputs of the software are mainly simulation and the report for the climbing activity program. The software should allow the user to wander around the 3D visualization of target area. At the same time the simulation will inform the users about route, timings, camp places, and weather conditions.

All the problems stated above should be solved to get a final product.

1.2 Project Scope

The software will have 3 main functionalities. First functionality is the input gathering. This functionality both requires user interaction and web/database connectivity. Second and the most important functionality is the planning. This one will plan the activity according to given information. Details about how this functionality use this information is described in following parts. The last one is the simulation and reporting. After user, geographic and weather data entered to the system and planning is over, the system will return simulation of route and the target area to the user or a report including activity program.

2. Design Constraints

2.1 Resource Constraints

We want to use most convenient technologies to achieve our aim. We tested and used some of GIS development kits such as GRASS, OpenGIS, GeoTools. These development kits are similar but the main difference shows up when it comes to configuring them. We tried to use GRASS first. In the official site of GRASS, it is described as:

"Commonly referred to as GRASS, this is a Geographic Information System (GIS) used for geospatial data management and analysis, image processing, graphics/maps production, spatial modeling, and visualization."

And GeoTools:

"Geo Tools is an open source (LGPL) Java code library which provides standards compliant methods for the manipulation of geospatial data, for example to implement Geographic Information Systems (GIS)."

Although GRASS claims that it can be natively developed on Windows platforms, it is very hard to configure on a Windows system. It takes almost 30 steps to have a working GRASS platform. On the other hand, GeoTools have a user friendly and easy to understand documentation and tutorials. Besides it is easily configured for Windows systems. As stated in the official definition, GeoTools is a Java code library. As a consequence we will implement our solutions in Java. GeoTools has some limitations: It doesn't support JDK 1.6 for now so we will use JDK 1.5. Also we stated before that we will use Netbeans but we changed our minds after using Eclipse.

Eclipse support for GeoTools is better than Netbeans. Maven is another tool we will use in software development process. Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. We will use *'Subversion'* as CVS tool. Subversion is an open source version control system. Using it, we can record the history of sources files, and documents. For holding user data we will use MySQL and for gathering data from web for weather data we will use JAXP (Java XML Processing API). All the other things like reporting, simulation and reporting we will use again Java APIs.

2.2 Time Constraints

The software development process has strict deadlines. We will obey these deadlines to achieve a successful result. After design phase, in the programming phase we need to allocate time for testing. Testing will have a big role for a good product.

2.3 Integrity Constraints

As we stated before, testing have a big role in development process and for a good integrity of modules we will test them separately. Some modules need completion of other modules so we will focus on independent modules first. Hierarchically we will continue to develop modules. After we get modules working correctly, we will start to integrate them to system. We aim not to lose time in this way.

2.4 Performance Constraints

The software should give result in a reasonable time. This reasonable time is mainly affected by planning process. We need to find a quick algorithm to find correct route considering external conditions. Performance of gathering these external conditions is affected by web connection speed and database design. After the planning process has been completed we need to show the resulting path and map to the user without using too much system resources. Our aim is to provide user with correct result(s) in minimum time using minimum system resources.

3. System Modules

The system designed for climbers will have a user friendly interface. Via this interface user will enter the data for defining activity plan. This data can be predefined. The database will hold

defined groups and users. A moderator will have right to access the database and choose users to create a new group or edit an existing group.

User will have options for some of the inputs. After the required areas are filled, according to data given to system, web connection will start and the system will get regarding information from Web Mapping Service, Web Featuring Service, Web Covering Service and weather service. All the data gathered from user and web will be used in planning phase. Output of the planning phase will be activity plan(s) for group. Details of activity plan are explained in following parts. After planning the route user will see the simulation of the activity and 3D visualization of map. The simulation that is showed to the user is done by another system component. If the user wants, the report of the route for activity will be printed or saved in a desired type.

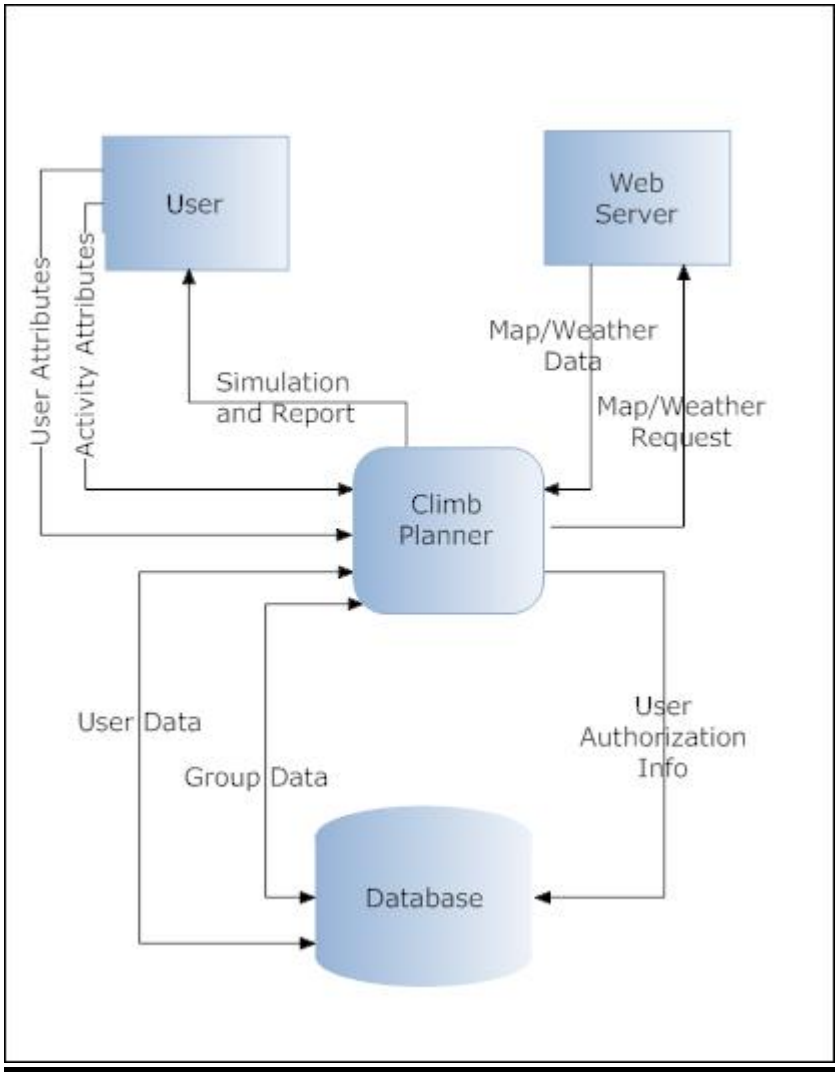


Figure 1: Level0 Data Flow Diagram of ClimbPlanner

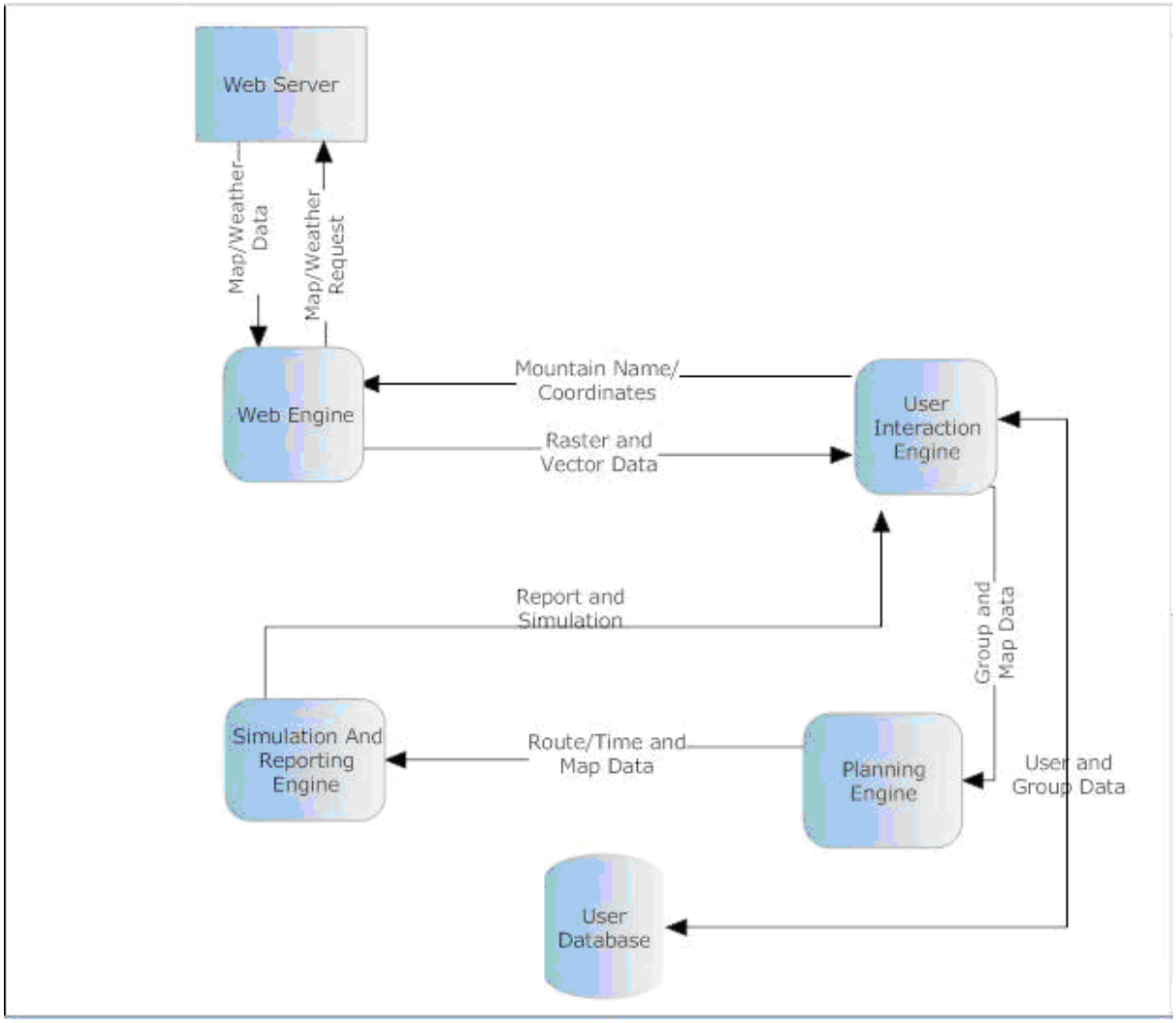


Figure 2: Level1 Data Flow Diagram for ClimbPlanner

After giving a very shallow explanation about system, detailed information about system modules is given below. There are 4 main components composing the system.

3.1 User Interaction Engine:

This engine is the only way of communicating with the user. We will design a user friendly interface. *User Interaction Engine* will have 4 sub-modules apparently. First sub-module, namely *Data Input*, will accept the data from user. Second sub-module, *Visual Data Representation*, will show the 3D visualization of map to the user. Third sub-module will do the *User Authentication* and last sub-module is a transfer module between *Simulation & Reporting Engine* and user.

3.1.1. Sub-Modules

3.1.1.1 Data Input:

We have two groups of users: Moderators and users. Moderators are also kind of users but their difference is they have right to do some privileged operations. The operations can be listed as:

1. Adding User
2. Editing User*
3. Deleting User*
4. Creating Group
5. Editing Group
6. Saving Group
7. Deleting Group
8. Deleting User from a Group
9. Adding User to a Group
10. Changing Password*
11. Adding Moderator
12. Searching User

**: Users have right to do this operation.*

Moreover planning a new activity requires moderator authorization. All the users will have a username and password stored in database. Since activity planner is moderator, moderator will provide the system with map data. The system gets map data in 2 ways; whether the map files or the coordinates/name of the target area. *User Interaction Engine* will react accordingly: If the map data given is a map file then this file will be sent to *Planning Engine.* Else if the map data given is coordinate/name of the target area, then this data will be sent to *Web Engine.*

In addition to this information, another data like beginning date of activity, type of report, simulation requests, 3D visualization requests, specifications about the route and user data will be

entered to the system via *User Interaction Engine*. The detailed information about which data that users enter to system can be found in database design section. But we need to state about equipment conditions here because user will choose his/her equipments from a list we provide. We intended to standardize equipment in this way. The list we provide will consist of equipments that are used widely.

- Carabineers
- Crampons
    - Ice Crampons
    - Rock Crampons
- Harness
- Rock Shoes
- Boots
- Supplies like camping cylinder, first-aid kit
- Backpack
- Tent
- Sleeping Bag
- Ropes
- Chock nuts
- Parkas

User will choose the equipment he/she has and then system will keep them in the database. Users can edit any time their equipment condition and other characteristics but in addition to that moderators are also allowed to change the experience points of users.

3.1.1.2. 3D Visualization

In this part users will be able to see their activity area in 3D. They will also be able to rotate, pan, tilt and zoom to this display. GeoTools have these features built-in. We will use these

functionalities of GeoTools. ClimbPlanner just call the built-in functions of GeoTools and user will see desired results on visualization window. The map to visualize will be obtained from *Planning Engine*. In this display users will not see their desired route. Main functionality of this sub-module is only showing users the area of activity in detail and allowing them to wander target area.

3.1.1.3 User Authentication

The purpose of this sub-module is allowing users to access the system and to determine which group (moderator or user) the user belongs to. The username and password will be compared with the username and the password stored in database. Also assigning each user with a password will prevent other users to edit or delete attributes of other users. After this authentication is completed user will be able to edit his/her settings or if the user is moderator he/she will be able to make an activity plan. According to result of authentication phase, only moderators will be able to do privileged jobs defined in *User Input* sub-module and add new map data.

3.1.1.4 Simulation and Reporting

This sub-module will dependent on *Simulation & Reporting Engine*. User will interact with *Simulation & Reporting Engine* with stop, pause, rewind, play and forward buttons. Except these methods user will not have any connection to this engine. Another responsibility of this sub-module is allowing user to choose the format of report. The details of what are included in report are explained in the *Simulation & Reporting Engine* section. Report file can be saved or printed according to user's request. This request will be forwarded to *Simulation & Reporting Engine* again and this engine corresponds accordingly.

3.1.2 Dependencies

User Interaction Engine has dependency over all other engines *Planning Engine, Simulation & Reporting Engine* and *Web Engine*. The biggest dependency of *User Interaction Engine* is over the *Simulation & Reporting Engine*. All the user requests like rewinding, playing, fast forwarding, pausing and stopping the simulation are redirected to *Simulation & Reporting Engine* and results are shown to users. *Simulation and Reporting* sub-module of the *User Interaction Engine* has nothing to do except to show simulation to the user and redirecting user requests to *Simulation &*

Reporting Engine. But we want engines to do maximum job without too big engine sizes, we decided to design *Simulation & Reporting Engine* separately from *User Interaction Engine*.

3D Visualization sub-module of *User Interaction Engine* will have dependency on *Planning Engine* since it does not matter the given map data is a map file or not, *User Interaction Engine* will obtain the map file from *Planning Engine*. The performance of this sub-module will be mainly affected by the performance of *Web Engine* and *Planning Engine*.

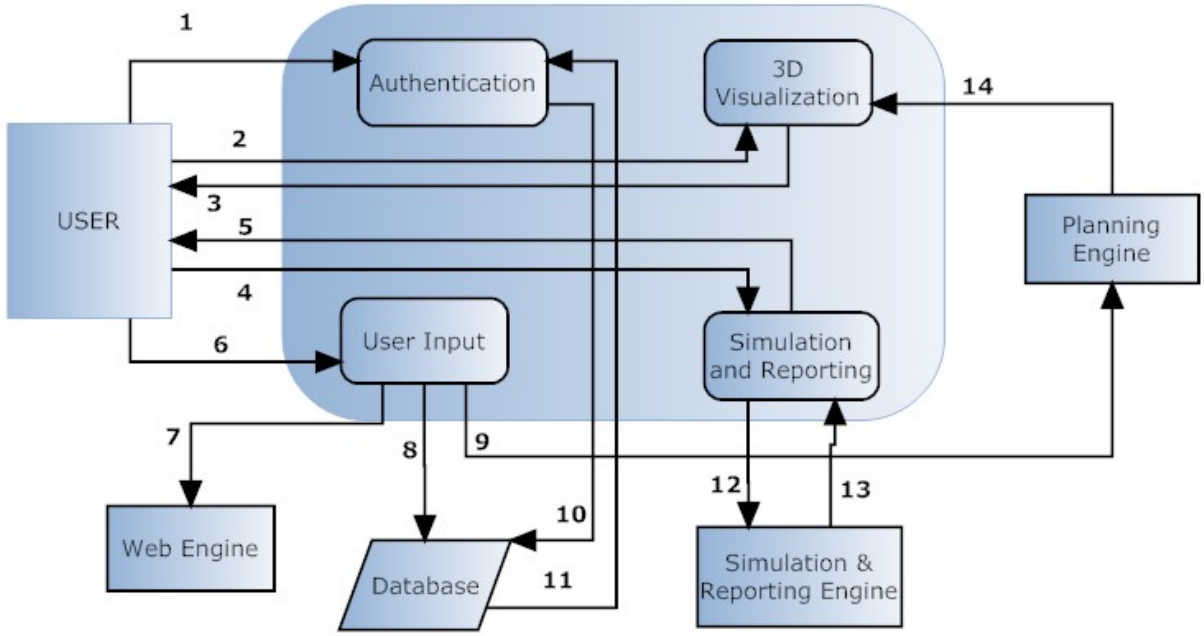


Figure 3:Level2 Data Flow Diagram for User Interaction Engine

Input and Output Specifications:

1. *User Name and Password*
   - ✓ *java.lang.String*
2. *Rotation, Spin, Tilt, Zoom, Pan Requests*
   - ✓ *java.lang.Integer*
3. *Returning display for 2*
   - ✓ *Required layers and features of Vector and Raster Map files*

4. *Requests for report and simulation*
    - ✓ *Requests for Report will be java.lang.Integer Pair Data Structure, first value for saving or printing document and second for type of document*
    - ✓ *Requests for Simulation java.lang.Integer*
5. *Return Data for 4*
    - ✓ *A document for reporting*
    - ✓ *Simulated data for simulation*
6. *Two types user input*
    - ✓ *Details about user attributes defined in Database section*
    - ✓ *Map data have 3 options*
        - *i) Name -> java.lang.String*
        - *ii) Coordinate -> java.lang.Double Data Structure*
        - *iii) Map file -> Defined in GIS File Formats*
7. *Map Data*
    - ✓ *Name -> java.lang.String*
    - ✓ *Coordinate -> java.lang.Double Pair Data Structure*
8. *Details about user attributes defined in Database section*
9. *Two data types*
    - ✓ *Group id -> java.lang.Integer*
    - ✓ *For beginning data of the activity -> java.util.Date*
    - ✓ *For route options ->java.lang.Integer*
10. *User Password Request*
11. *Username and Password pair*

12. Two different data to be sent

- ✓ *Reporting*
    - *i) Java.lang.Integer Pair*
- ✓ *Simulation*
    - *i) Java.lang.Integer*

13. Approval for Requests

14. Map file with raster data set

- ✓ *Required layers and features of Vector and Raster Map files*

3.2 Planning Engine:

This engine may be the most important engine of the system. Almost all the other engines have dependency over this engine. Main functionality of this engine is producing an accurate route using external conditions like activity related requests, group data, geographic data and weather data. The detailed list of external conditions can be listed as:

Activity Related Requests:

- Shortest Path to the Peak
- A Path that Requires Shortest Time
- Safest Path to the Peak
- A Path That Requires Minimum Equipment

Group Data:

Groups will consist of members and group characteristics will be based on user characteristics. User characteristics are:

- Physical Condition
    - o Weight
    - o Age

- Gender
- Required Resting Time
- Maximum Walking/Climbing Time without Break
- Maximum Walking/Climbing Distance without Break
- Maximum Load User can Carry
- Equipment Condition
- Speed Characteristics of User on Different Surfaces and Weather Conditions,
    - On Rock,
    - On Ice,
    - On Soil,
    - On Snow,
    - On Rain,
    - On Storm,

The system behaves the group like it is a user. The group data will hold the most deficient characteristics of the members. If the slowest member of the group has a speed of 1 km/h then the speed of the group will be 1 km/h. Or one member has no ice crampons then system will behave like nobody has ice crampons.

Geographic Data:

This data will be used to plan a route considering group data and user requests stated above. We will use raster and vector data of the target area. We will get elevation data from raster data and geographic characteristics of the area from the vector data. These characteristics are the terrestrial shapes that group can encounter with during the climb:

- Rivers,
- Rocks,
- Icy Surfaces,

- Ponds,
- Quicksand, Swamps,

Weather Data:

For some seasons some routes may be closed for climbing or too dangerous to climb. Moreover, even if the route is safe enough to climb on a sunny day, raining could affect the quality or the safety of the activity. The system must consider these parameters before giving a result. Finding exact weather estimations for all possible climbing areas may not be possible but we want to give estimations about the region of the climbing area as much as possible. Using this information, users will be able to reconsider their activity date. Also the weather conditions have a big affect on their speeds on climbing. Our program will calculate the route keeping in view whether it will be raining, snowing, there will be a storm or fog during the activity. On Simulation and Reporting sub-module of the *User Interaction Engine* users will be informed about weather forecasts with small icons.

3.2.1 Sub-Modules

In the design of *Planning Engine*, we have 2 sub-modules. First one will be about route planning generally and the other one is about comparisons and qualification of requirements of the calculated routes and user capabilities.

3.2.1.1. Route Planner

Main functionality of this sub-module is to calculate routes for given beginning and ending points considering group capabilities. Input of this module will be map data and group data and activity related requests. This module will calculate the routes according to given beginning and ending points and send the resulting routes to *Comparison and Qualification* sub-module.

Using elevation information and terrestrial characteristics of the area that are found in map data, our algorithm will try to find next points of the route one by one. During this process only some routes which are appropriate for the group will be qualified for the other sub-module. During this process system will hold the requirements like equipment and experience information of each step in a data structure. Using weather information is not required here because we don't intend to calculate time issues of climbing. At the end of the process of this sub-module we may have more than one route and all of these routes are appropriate for climbing group. Together with this, if the program will not be able to find a suitable route then users will be informed about that.

As we stated above, requirements of each route will be kept in a data structure and this data structure will be used by *Comparison and Qualification* sub-module. Outputs of *Route Planner* sub-module will be the routes in means of coordinates and a data structure holding requirements for each route. Specifications about inputs and outputs are at the end of *Planning Engine* section.

3.2.1.2. Comparison and Qualification

As mentioned above, this module will be responsible for calculation of the safest route, the route with the shortest distance to the peak, the route with the shortest time to the peak, the route that requires minimum equipment etc. Additionally, this sub-module will calculate camping locations according to the inputs such as maximum walking time/distance without break, maximum climbing time/distance without break from user and the inputs like geographical conditions from map data. This sub-module will consider mainly 4 things to decide on camping places. First, camping place must be a horizontal area. We will use elevation data of the map and determine camping places that have a slope small enough. Considering only slope will not be enough for a place to be a camping place so we will consider the distance between previous camping place and the next one. This distance can not be greater than any user's *'Maximum distance taken without break'* value. Moreover the time amounts between adjacent camp places can not be longer than any user's *'maximum climbing time without break'* and *'maximum walking time without break'* values. For groups that have a lot of members the camping place must be wide enough, too. Taking terrestrial data into consideration, this sub-module will also calculate the required area for camping for groups.

The inputs and outputs of this sub-module are defined technically at the end of *Planning Engine* section.

3.2.2. Dependencies

In order to make *Planning Engine* work correctly, it needs some inputs from other engines. These engines are *User Interaction Engine* and *Web Engine*. Inputs of *Planning Engine* are mainly activity and user related data, geographic and weather data. Activity related data stated above directly come from *User Interaction Engine* and user related data come from *User Interaction Engine* or database. *User Interaction Engine* must transfer data correctly to *Planning Engine*. It means that all the required data for planning must be gathered in the *User Interaction Engine*. The data coming from *User Interaction Engine* will mainly be used by *Comparison and Qualification* sub-module of *Planning Engine*. For predefined users and groups, *Planning Engine* will have to

connect to database to get required data. The design of database may affect the performance of *Planning Engine*. Moreover, if user gives the map data in the form of a file, this data again will be used in *Comparison and Qualification* sub-module. In other cases *Web Engine* will be responsible for supplying map data. This dependency also results in performance issues. Speed of internet connection affects *Web Engine*'s performance and hierarchically performance of *Planning Engine* is also affected by the performance of *Web Engine*.

Second sub-module, namely *Route Planner*, is dependent on map data, group capabilities and two coordinates –beginning and ending points of activity-. This map data is taken from either user or *Web Engine*. The situation of performance dependency over *Web Engine* exists here, too. Although there is some dependency to *User Interaction Engine*, it is very small compared to dependency over *Web Engine*. Additionally, comparisons between user capabilities and requirements of the path are another performance related issue.

Interdependencies between 2 sub-modules are also important enough to specify. *Route Planner* sub-module does not require any result of *Comparison and Qualification* sub-module to do its task but the reverse is valid. *Comparison and Qualification* sub-module will need some routes to process and some route requirements to compare the values of user requests and these requirements. To get correct results from *Planning Engine*, two sub-modules need to work correctly and synchronously.

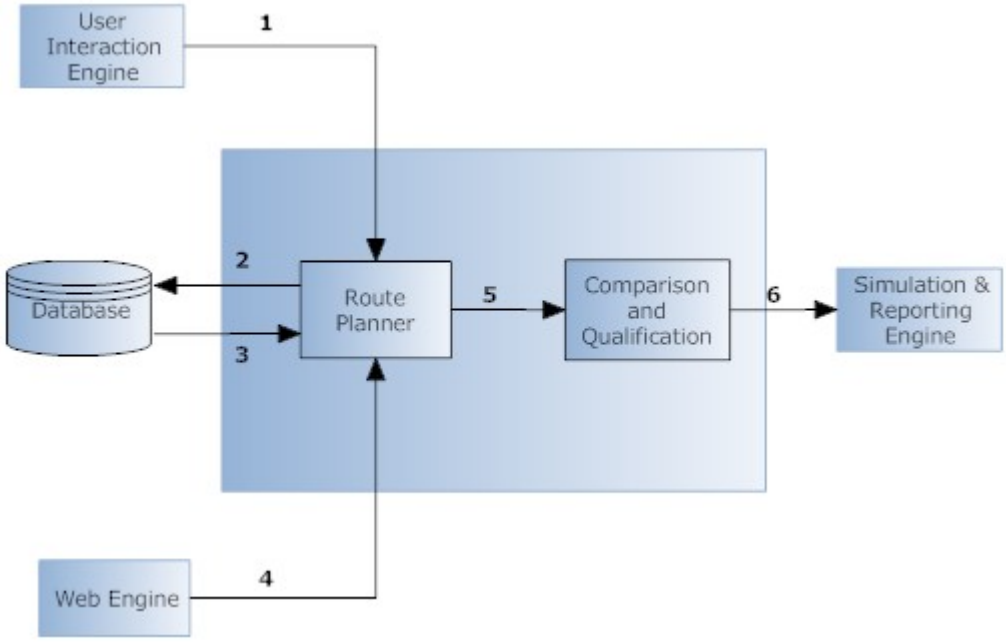


Figure 4: Level2 Data Flow Diagram for Planning Engine

Input and Output Specifications:

1. *Two data types*
   - ✓ *Group id -> java.lang.Integer*
   - ✓ *For beginning date of the activity -> java.util.Date*
   - ✓ *For route options -> java.lang.Integer*
2. *Group id -> java.lang.Integer*
3. *Attributes of users of group, details about this data are explained in database section*
4. *Vector and Raster Map Files and Weather data*
   - ✓ *GeoTIFF, DTED map files*
   - ✓ *An XML file that includes metadata about weather estimations*
5. *Two data input to be sent*
   - ✓ *A data structure that holds routes*
   - ✓ *Map files*
   - ✓ *Weather information*
6. *A data structure that holds coordinates of route with attributes that are camping places and timings*

3.3 Simulation & Reporting Engine:

Main purposes of this engine are to simulate the route calculated and printing/saving a report about activity plan including route, projection of map, timings and group information. As we mentioned before this engine could be included in *User Interaction Engine* but we don't want to design too big engines doing irrelevant things together.

Inputs of this engine are the weather data, map data, route data and user requests. Weather data and map data are already used in *Planning Engine*. This information again will be obtained from *Planning Engine*. Also route information will be sent by *Planning Engine* to *Simulation & Reporting Engine*. User requests will be sent via *User Interaction Engine*. These requests are

playing, stopping, rewinding, fast forwarding, pausing and replaying the simulation. Except these requests user will not have any interaction with simulation. We are also planning user to be able to get a video output from simulation but for now it's an extra feature and we are researching about it.

3.3.1. Sub-Modules

There are 2 main modules of *Simulation & Reporting Engine:*

3.3.1.1. Simulation

Main functionality of this unit is to show users simulation of the route calculated in the *Planning Engine*. We are planning to use Java 3D API to achieve this purpose. There are 3 main parts of the data to be simulated. First part will be the route, the second part will be the map data and the last part will be weather data. Route information will be requested from *Planning Engine* and it will be depicted over the map. Although, this process will require some 3D operations over the map, the map data will not be manipulated by the program. *The Simulation* sub-module will only show the route on the mountain. Simulation will show the route as an extending arrow from the starting point to the end point of the climbing activity. On this extending arrow, camp places and timings of distance taken will be shown to the user. Weather data is the fundamental data that will be presented in this step because the user should have a visual weather condition of the region which is obtained from an RSS feed or XHTML version of weather service and will be demonstrated with a small picture such as sunny, rainy, foggy, windy etc. in one corner of the simulation window.

The simulation sub-module will directly send the simulated data to the *Simulation and Reporting* sub-module of the *User Interaction Engine.*

3.3.1.2. Reporting

Main responsibility of this sub-module is to send the route information and map data projection to a printer or to save as a desired type. Inputs of this sub-module will be user requests like type of the report document, map data and route information sent from *Planning Engine*. We need to add route to the 2D projection of map and then save or print it. Together with route and map information, the users that are attending climbing activity will be listed in the report file. The entities which are included in the report file are listed below:

- Name of the activity group

- Name of the planner of the activity
- Names of the members attending activity
- Image of the area that will be used for activity
- Weather information for the activity
- Route information for the activity
- Required equipment list for the activity
- Beginning date of activity
- Camping places and timings for activity

Details about the inputs and outputs of these sub-modules will explained at the end of *Simulation & Reporting Engine.*

3.3.2. Dependencies

The main purpose of the engine is to simulate or report the pre-compiled data so there are some dependencies over other engines. Making *Simulation & Reporting Engine* work correctly, requires *Planning Engine* to do its job. Simulation sub-module needs correct route information from *Planning Engine*. Moreover, the map and weather information that are shown to user in simulation are also requested from *Planning Engine*. *Planning Engine* must guarantee that map data and weather data are not manipulated during the planning phase. Since, the map data used in *Planning Engine* is requested from *Web Engine* and after planning it is transferred to *Simulation Engine*, the performance of *Simulation Engine* is dependent to performance of these 2 engines.

Reporting sub-module of *Simulation & Reporting Engine* will need information about the activity and group. Activity information includes date of the beginning of activity, map data, weather data and route. Date of the beginning of activity is requested from *User Interaction Engine.* The other required information will be obtained from *Planning Engine.* For requested document type to be printed or saved and group information, reporting sub-module will need to connect to database and *User Interaction Engine*. Information about members of activity group will be requested from database. *User Interaction Engine* will send an identification of the group that needs an activity plan. This identification will be used in printing or saving the report of the

activity. Database design and performances of *Web Engine*, *Planning Engine* and *User Interaction Engine* have a role over the performance of reporting sub-module.

There is no dependency between two sub-modules of *Simulation & Reporting Engine.*

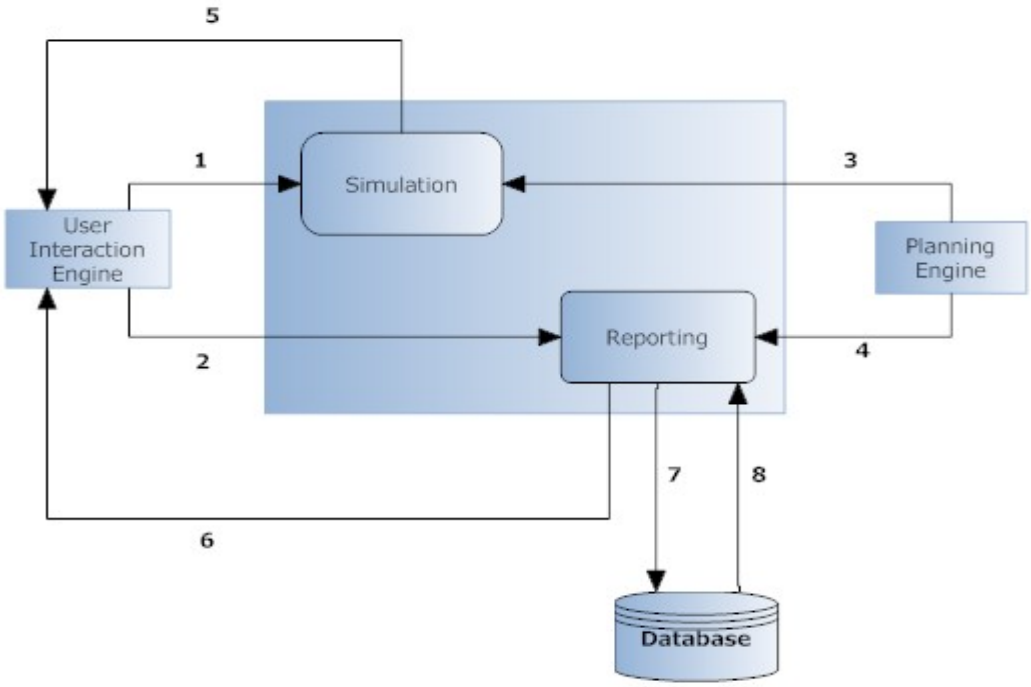


Figure 5: Level2 Data Flow Diagram for Simulation & Reporting Engine

3.3.3 Input and Output Specifications

1. *Requests regarding Simulation*
   - ✓ *Java.lang.Integer*
2. *Requests regarding Reporting*
   - ✓ *Java.lang.Integer Pair*
3. *Data to be processed for simulation*
   - ✓ *Map files -> DTED, GeoTIFF*
   - ✓ *Weather Data -> Java.lang.Double for Temperature and Java.lang.Integer for precipitation situation.*
   - ✓ *A data structure that holds coordinates(Java.lang.Float) of route with attributes that are camping places and timings*

4. *Data to be processed for report*
    - ✓ *Map projection of 2D map data*
    - ✓ *A data structure that holds coordinates(Java.lang.Float) of route with attributes that are camping places and timings*
    - ✓ *Group id*
5. *Visual data of simulation*
6. *Approval of requests*
7. *Group id -> Java.lang.Integer*
8. *Attributes of users that belongs to the group with the group id.*

3.4 Web Engine:

The working mechanism of *Web Engine* is not too complex but it requires a lot of task to do if it is requested by user. We need to connect some map data servers and these servers must be capable of giving data about mainly elevation data, coordinate system descriptions and terrestrial shapes of the earth. Map data we refer frequently is the data we obtain from web servers. *Web Engine* will use Web Featuring Service (WFS), Web Mapping Service (WMS), and Web Coverage Service (WCS) to accomplish its task. GeoTools supports connection with these services and the tasks that our system must do will be done mainly over GeoTools' built-in connectors.

In addition, weather data will be obtained from a web server. The process schema of this data mining is below:

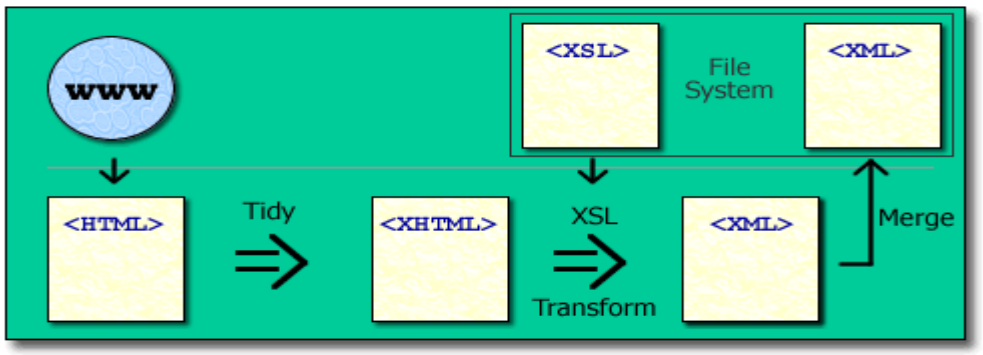


Figure 6: Working mechanism of gathering data from web server

The steps are listed here to give a brief overview of the process:

- Identify the data source and map it to XHTML.
- Find reference points within the data.
- Map the data to XML.
- Merge the results and process the data.

We are planning to use Java API for XML Processing (JAXP) to get and parse XML data obtained from web. In the official site of JAXP, it says:

"The Java API for XML Processing (JAXP) enables applications to parse, transform, validate and query XML documents using an API that is independent of a particular XML processor implementation. JAXP provides a pluggability layer to enable vendors to provide their own implementations without introducing dependencies in application code. Using this software, application and tool developers can build fully-functional XML-enabled Java applications for e-commerce, application integration, and web publishing."

We need data formats that include elevation data, coordinate system descriptions and terrestrial shapes of the earth. These data will be used in *Planning Engine* and *Simulation & Reporting Engine*.

Before explanation about *Web Engine* it is required to give some background information about GIS map file formats and web services.

GIS File Formats:

A GIS file format is a standard of encoding geographical information into a file. They are created mainly by government mapping agencies (such as the USGS) or by GIS software developers.

Metadata often includes:

- *Elevation data, either in raster or vector (e.g., contour lines) form*
- *Shape layers, usually expressed as line drawings, for streets, postal zone boundaries, etc.*
- *Coordinate system descriptions.*
- *One or more data describing the precise shape of the Earth assumed by the coordinates.*

Raster Data Types:

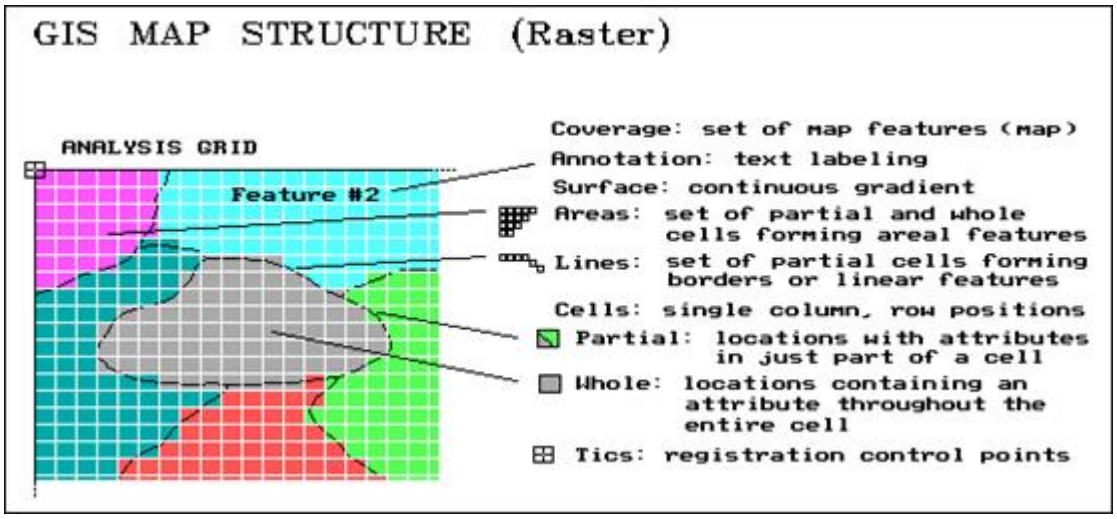


Figure 7: Raster Map Data Structure

- *ADRG - National Imagery and Mapping Agency (NIMA)'s ARC Digitized Raster Graphics*
- *BIL - Band Interleaved by Line (image format linked with satellite derived imagery)*
- *CADRG - National Imagery and Mapping Agency (NIMA)'s Compressed ARC Digitised Raster Graphics (nominal compression of 55:1 over ADRG)*
- *CIB - National Imagery and Mapping Agency (NIMA)'s Controlled Image Base (type of Raster Product Format)*
- *Digital raster graphic (DRG) - digital scan of a paper USGS topographic map*
- *ECW - Enhanced Compressed Wavelet (from ERMapper). A compressed wavelet format, often lossy.*
- *ESRI grid - binary and ASCII raster formats used by ESRI*
- *GeoTIFF - TIFF variant enriched with GIS relevant metadata*
- *IMG - ERDAS IMAGINE image file format*
- *MrSID - Multi-Resolution Seamless Image Database (by Lizardtech). A compressed wavelet format, often lossy.*
- *JPEG2000 - Open-source raster format. A compressed format, allows both lossy and lossless compression.*

Vector Data Types

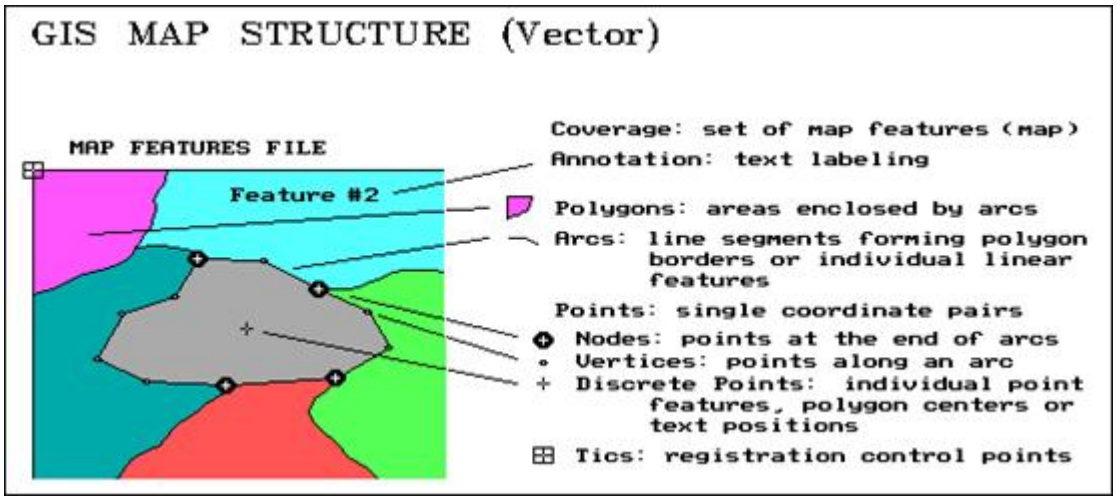


Figure 8: Vector Map Data Structure

- *Geography Markup Language (GML) - XML based open standard (by OpenGIS) for GIS data exchange*
- *DXF - Contour elevation plots in AutoCAD DXF format*
- *Shapefile - ESRI's open, hybrid vector data format using SHP, SHX and DBF files*
- *Simple Features - Open Geospatial Consortium specification for vector data*
- *MapInfo TAB format - MapInfo's vector data format using TAB, DAT, ID and MAP files*
- *National Transfer Format (NTF) - National Transfer Format (mostly used by the UK Ordnance Survey)*
- *TIGER - Topologically Integrated Geographic Encoding and Referencing*
- *Cartesian coordinate system (XYZ) - Simple point cloud*
- *Vector Product Format - National Imagery and Mapping Agency (NIMA)'s format of vectored data for large geographic databases.*
- *GeoMedia - Intergraph's Microsoft Access based format for spatial vector storage.*
- *ISFC - Intergraph's MicroStation based CAD solution attaching vector elements to a relational Microsoft Access database*
- *Personal Geodatabase - ESRI's closed, integrated vector data storage strategy using Microsoft's Access MDB format*
- *File Geodatabase - ESRI's geodatabase format, stored as folders in a file system.*
- *Coverage - ESRI's closed hybrid vector data storage strategy. Legacy ArcGIS Workstation / ArcInfo format with reduced support in ArcGIS Desktop lineup*

Grid Data Types (for elevation)

- *USGS DEM - The USGS' Digital Elevation Model*

- *DTED - National Imagery and Mapping Agency (NIMA)'s Digital Terrain Elevation Data*
- *GTOPO30 - Large complete Earth elevation model at 30 arc seconds*
- *SDTS - The USGS' successor to DEM*

(http://en.wikipedia.org/wiki/GIS_file_formats)

Advantages and Disadvantages of Vector and Raster Data Types

Vector Data

Advantages:

- Data can be represented at its original resolution and form without generalization.
- Graphic output is usually more aesthetically pleasing (traditional cartographic representation);
- Since most data, e.g. hard copy maps, is in vector form no data conversion is required.
- Accurate geographic location of data is maintained.
- Allows for efficient encoding of topology, and as a result more efficient operations that require topological information, e.g. proximity, network analysis.

Disadvantages:

- The location of each vertex needs to be stored explicitly.
- For effective analysis, vector data must be converted into a topological structure. This is often processing intensive and usually requires extensive data cleaning. As well, topology is static, and any updating or editing of the vector data requires re-building of the topology.
- Algorithms for manipulative and analysis functions are complex and may be processing intensive. Often, this inherently limits the functionality for large data sets, e.g. a large number of features.
- Continuous data, such as elevation data, is not effectively represented in vector form. Usually substantial data generalization or interpolation is required for these data layers.
- Spatial analysis and filtering within polygons is impossible

Raster Data

Advantages:

- The geographic location of each cell is implied by its position in the cell matrix.

- Accordingly, other than an origin point, e.g. bottom left corner, no geographic coordinates are stored.

- Due to the nature of the data storage technique data analysis is usually easy to program and quick to perform.

- The inherent nature of raster maps, e.g. one attribute maps, is ideally suited for mathematical modeling and quantitative analysis.

- Discrete data, e.g. forestry stands, is accommodated equally well as continuous data, e.g. elevation data, and facilitates the integrating of the two data types.

- Grid-cell systems are very compatible with raster-based output devices, e.g. electrostatic plotters, graphic terminals.

Disadvantages:

- The cell size determines the resolution at which the data is represented.

- It is especially difficult to adequately represent linear features depending on the cell resolution. Accordingly, network linkages are difficult to establish.

- Processing of associated attribute data may be cumbersome if large amounts of data exist. Raster maps inherently reflect only one attribute or characteristic for an area.

- Since most input data is in vector form, data must undergo vector-to-raster conversion. Besides increased processing requirements this may introduce data integrity concerns due to generalization and choice of inappropriate cell size.

- Most output maps from grid-cell systems do not conform to high-quality cartographic needs.

We need to use each data format's advantage to get desired result from engines. For example, the vector data model does not handle continuous data, e.g. elevation, very well while the raster

data model is more ideally suited for this type of analysis. Accordingly, the raster structure does not handle linear data analysis, e.g. shortest path, very well while vector systems do.

Web Services:

Web Mapping Service:

The Open Geospatial Consortium, Inc. (OGC) Web Map Service (WMS) specification is an international specification for serving and consuming dynamic maps on the Web. WMS services are useful if you want to make your maps available online in an open, recognized way across different platforms and clients. Any client built to support the WMS specification can view and work with your service. Four versions of the WMS specification have been published so far. They are v1.0.0, v1.1.0, v1.1.1, and v1.3.0 (most recent).

Client applications work with a WMS service by appending parameters to the service's URL. WMS services published with almost all the servers support the following three operations:

- Request metadata about the service (GetCapabilities)
- Request a map image (GetMap)
- Request information about features in the map (GetFeatureInfo (optional))

It is not necessary for a WMS service to support all the operations. But one must support at least GetCapabilities and GetMap operations to be a "Basic WMS", and support optional GetFeatureInfo operation to be a "Queryable WMS".

The maps returned by a WMS service are images only. They do not contain actual data. To expose map data as vector features through OGC specifications, publish a WFS service instead. To expose map data as raster layers, publish a WCS service.

The online resource of each operation supported by a compliant WMS server is an HTTP Uniform Resource Locator (URL), so a WMS service can be considered as a Representational State Transfer (REST) service. Unlike a standard Web Service, a SOAP client is not necessary for consuming a WMS service, and a Web browser is the simplest client. You can get a WMS service's service level metadata, a map image, or attributes values of a feature all by sending a URL request to the server and viewing the corresponding responses in browser either as an XML document or an image.

	Mandatory / Optional	Default value	Description
VERSION / WMTVER	Optional	"1.3.0" (9.3) "1.1.1" (9.2)	Request version. Only use "WMTVER" in WMS 1.0.0. Otherwise use "VERSION".
SERVICE	Mandatory	N/A	Service type. It must be set to "WMS". This parameter is not available in WMS 1.0.0
REQUEST	Mandatory	N/A	Request name. It must be set to "GetCapabilities". For WMS 1.0.0, it must be set to "capabilities".
FORMAT	Optional	"text/xml" (9.3)	Output format for service metadata. Only available in WMS 1.3.0. Example: "application/vnd.ogc.wms_xml", "text/plain" or "text/html".

Figure 9: Parameters of a WMS GetCapabilities request URL

	Mandatory / Optional	Default value	Description
VERSION / WMTVER	Mandatory	N/A	Request version. Only use "WMTVER" in WMS 1.0.0. Otherwise use "VERSION".
SERVICE	Mandatory	N/A	Service type. It must be set to "WMS". This parameter is not available in WMS 1.0.0
REQUEST	Mandatory	N/A	Request name. It must be set to "GetMap". For WMS 1.0.0, it must be set to "map".
LAYERS	Mandatory	N/A	Comma-separated list of one or more map layers. Example: "0,1,2", "layer1,layer2"
STYLES	Mandatory	N/A	Comma separated list of one or more styles (one style per requested layer). Example: "0,1,2", "style1,style2"
CRS / SRS	Mandatory	N/A	Coordinate reference system. "CRS" is only used in WMS 1.3.0. Otherwise use "SRS". Example: "EPSG:4326", "EPSG:4269"
BBOX	Mandatory	N/A	Bounding box corners (lower left and upper right) in CRS/SRS unit. Example: "-180,-90,180,90".
WIDTH	Mandatory	N/A	Width in pixels of map picture. Example: "1024", "800".
HEIGHT	Mandatory	N/A	Height in pixels of map picture. Example: "768", "600".
FORMAT	Mandatory	N/A	Output format of map. Example: "image/png", "image/jpeg", "image/svg+xml".
TRANSPARENT	Optional	"False"	Background transparent or not. Example: "True", "False".
BGCOLOR	Optional	"0xFFFFFF"	Hexadecimal RGB color for the map background. Example: "0xFFFFFF", "FFFFFF".
SLD	Optional	" "	URL of Styled Layer Descriptor XML file.
EXCEPTIONS	Optional	"text/xml" (WMS 1.3.0) "application/vnd.ogc.se_xml" (WMS 1.1.1 and WMS 1.1.0) "application/vnd.ogc.inimage" (WMS 1.0.0)	The format in which exception should be reported. Example: "text/xml", "text/html".

Figure 9: Parameters of a WMS GetMap request URL

	Mandatory / Optional	Default value	Description
VERSION / WMTVER	Mandatory	N/A	Request version. Only use "WMTVER" in WMS 1.0.0. Otherwise use "VERSION".
SERVICE	Mandatory	N/A	Service type. It must be set to "WMS". This parameter is not available in WMS 1.0.0
REQUEST	Mandatory	N/A	Request name. It must be set to "GetFeatureInfo". For WMS 1.0.0 it must be set to "feature_info".
Map request part	Mandatory	N/A	Partial copy of the GetMap request parameters that generated the map for which information is desired.
QUERY_LAYERS	Mandatory	N/A	Comma-separated list of one or more map layers to be queried. Example: "0,1,2", "layer1,layer2"
INFO_FORMAT	Mandatory (WMS 1.3.0) Optional (WMS 1.1.1, 1.1.0, 1.0.0)	N/A (WMS 1.3.0) "text/xml" (WMS 1.1.1, 1.1.0, 1.0.0)	Return format of feature information. Example: "text/xml", "text/html".
I / X	Mandatory	N/A	X coordinate in pixels of feature in the map coordinate system. Example: "100", "200". Use "I" for WMS 1.3.0. Otherwise use "X".
J / Y	Mandatory	N/A	Y coordinate in pixels of feature in the map coordinate system. Example: "100", "200". Use "J" for WMS 1.3.0. Otherwise use "Y".
EXCEPTIONS	Optional	"text/xml" (WMS 1.3.0) "application/vnd.ogc.se_xml" (WMS 1.1.1 and WMS 1.1.0) "application/vnd.ogc.inimage" (WMS 1.0.0)	The format in which exceptions should be reported. Example: "text/xml", "text/html".

Figure 10: Parameters of a WMS GetFeatureInfo request URL

For each WMS server these parameters may be used in a URL in different ways but WMS servers react to these different URLs in the same way.

Web Featuring Service:

WFS is an open specification for serving geographic features over the Web. Unlike the OGC Web Map Service (WMS) that returns an image of a map, the WFS service returns actual features with geometry and attributes that clients can use in any type of geospatial analysis. WFS services also support filters that allow users to perform spatial and attribute queries on the data.

A Web browser is the simplest client of a WFS service. WFS requests can be issued through HTTP, and the responses or exceptions are returned through the browser. All WFS services support three operations: GetCapabilities, DescribeFeatureType, and GetFeature. Through URL parameters, you can use these operations to obtain service metadata, feature type information, and

GML-encoded features from the WFS service. These operations and parameters are detailed in the OGC WFS specifications.

Web Covering Service:

The Open Geospatial Consortium, Inc. (OGC) Web Coverage Service (WCS) provides an open specification for sharing raster datasets on the Web. The raster datasets made available through WCS services are coverages.

A WCS service returns data in a format that can be used as input for analysis and modeling. This is in contrast with OGC WMS services, which only return a picture of the data.

A WCS client can do three things with a WCS service:

- GetCapabilities--Returns service-level metadata and a brief description of the data collection
- DescribeCoverage--Returns a full description of one or more coverages
- GetCoverage--Returns a coverage in a well-known coverage format

WCS services must supply at least one of the GIS resources listed below:

- A map containing raster layers
- A raster dataset
- A layer file referencing a raster dataset
- A compiled image service definition
- A geodatabase

Each WCS service exposes service-level metadata through its capabilities file. The capabilities file is the XML response that clients receive when they make a GetCapabilities request on the service.

Supported output formats for WCS services are GeoTIFF, NITF, HDF, JPEG, JPEG2000, and PNG.

Web Engine will use these services to accomplish its task. There are many WMS, WFS and WCS servers to use but it is not important to choose one of them for now. The important thing is

here is sending the request for a map vie a valid URL for selected server and transferring the returned data to *Planning Engine.*

3.4.1 Sub-Modules

Since Web Engine has mainly two capabilities, it has two sub-modules:

3.4.1.1 Map Module

This module is responsible for providing raster and vector map data from the web servers. This module will take map name or map coordinates information from the *User Interaction Engine* and according to this information it will compose a valid URL to be sent to Web Services explained above.

In addition, it will be responsible for transferring the data returned from Web services to *Planning Engine*.

3.4.1.2 Weather Module

This sub-module will need to connect to a weather service like Yahoo! Weather and to do data mining from this weather service. The data returned from weather service will be an HTML file and after converting this HTML file to XML file; we will use Java XML Processing API (JAXP) to process it. After this process, temperature, speed of wind, precipitation situation will be sent to *Planning Engine*.

3.4.2 Dependencies

Web Engine will only dependent on *User Interaction Engine* because if the user has not an actual map file for the climbing area then it requires *Web Engine* to obtain it. User will give name or the coordinates for climbing area. Transferring the request from *User Interaction Engine* to *Web Engine* is not a big deal for performance. At the same time, transaction of map data and weather data may affect the performance of *Web Engine.*

Specifications about inputs and outputs for *Web Engine* are explained at the end of this section.

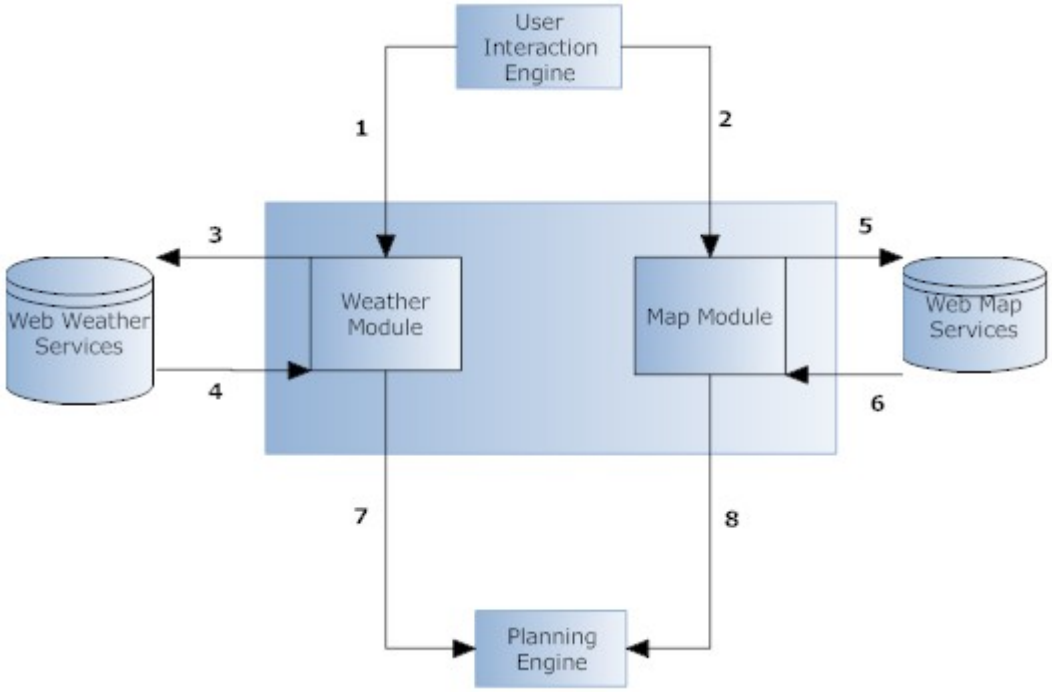


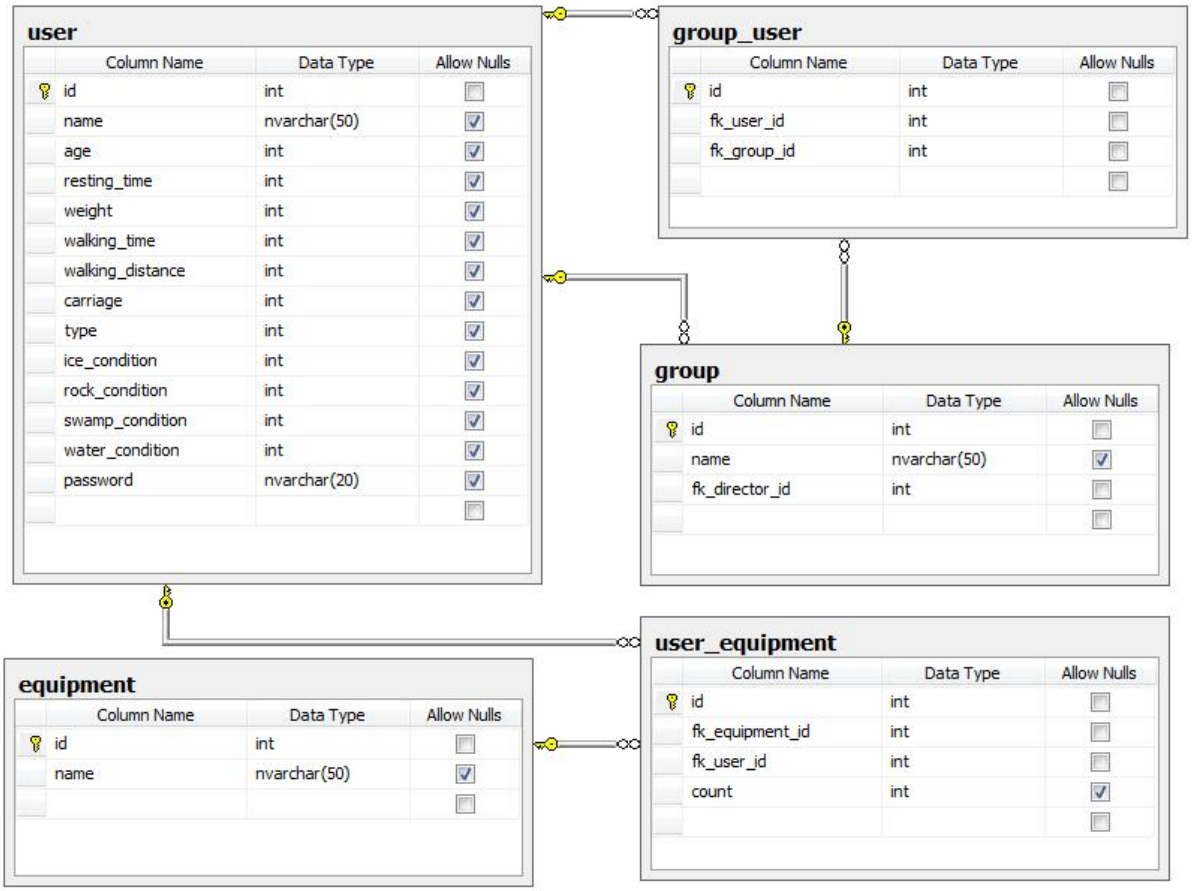
Figure 11: Level2 Data Flow Diagram for Web Engine

3.4.3 Input and Output Specification:

1. *Name or the coordinates of the area and the date of the activity*
   - ✓ *Java.lang.String or for coordinates Java.lang.Float*
   - ✓ *Java.util.Date for date*
2. *Name or the coordinates of the area*
3. *A valid URL for weather estimation of the area for the date given*
4. *An XML file*
5. *3 valid URLs for WCS,WMS and WFS*
6. *Map files -> GeoTIFF, DTED*
7. *Precipitation situation, temperature and wind speed*
8. *Map files -> GeoTIFF, DTED*

3.5 Database:

We hold user and group data in database. We will use MySQL for this purpose. In fact the data we will keep in database is not too complex. The diagram of the database design is as follows:



Explanations for each table's members are here:

User Table:

- id: For each member there will be a unique id
- name: Name of the user
- age: Age of the user
- resting_time: Minimum resting time of user
- weight: Weight of the user
- walking_time: Maximum walking time of user without break
- walking_distance: Maximum walking distance of user without break

- carriage: Maximum load that user can carry
- type: Type of the user either moderator or user
- ice_condition: The factor that affects user's speed and time characteristics on icy surface
- rock_condition: The factor that affects user's speed and time characteristics on rock surface
- swamp_condition: The factor that affects user's speed and time characteristics on swamp
- water_condition: The factor that affects user's speed and time characteristics in water
- password: Password of user

Group Table:

- id: For each group there will be a unique id
- name: Name of the group
- fk_director_id: id of the creator of group (foreign key pointing to id in User table)

User_Equipment Table:

- id: Identification of relation
- fk_equipment_id: Id of the equipment (foreign key pointing to id in Equipment table)
- fk_user_id: Id of the user (foreign key pointing to id in User table)
- count: Holds how many equipment with fk_equipment_id user has.

Equipment Table:

- id: Id of the equipment
- name: Name of the equipment

Group_User Table:

- id: Identification of relation
- fk_user_id: Id of the user which belongs to the group with id that is fk_group_id (foreign key pointing to id in User table)

- fk_group_id: Id of the group which the user -that has fk_user_id as its id- belongs to. (foreign key pointing to id in Group table)

4. Development Tools We Use:

In this part we explain the tools we use in development process.

4.1 Eclipse IDE:

In official site of Eclipse, it says:

"Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle."

We first intended to use Netbeans instead of Eclipse. Although Netbeans has Sun Microsystems assistance, when it comes to developing GIS technologies, we see that Eclipse has a big community and technical support over the web. As a consequence we decided to use Eclipse IDE. Also the tutorials and examples on the web related to GeoTools are all built with Eclipse. Eclipse has Maven, Subversion features, too.

http://www.eclipse.org/

4.2 Maven:

In official site of Maven, it says:

"Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information."

Maven is a Java deployment tool just like Ant. With Maven, developers can do whatever they can do with Ant. Maven has some better specialties like more efficient documentation and testing features compared to Ant. Also GeoTools supports Maven.

http://maven.apache.org/

4.3 Subversion:

Official definition of Subversion is:

"*Subversion is a free/open-source version control system. That is, Subversion manages files and directories, and the changes made to them, over time. This allows user to recover older versions of data, or examine the history of how data changed.*"

Because it is a must for this course and it is a good habit for programming to keep version history and changes across them to show our progress. Eclipse and GeoTools has Subversion support and because the most important factor for us is to have consistent development kits we decided to use Subversion.

http://subversion.tigris.org/

4.4 MySQL:

MySQL is the most popular open-source database management system. Eclipse database connection to MySQL is unproblematic. Also because it is open source and free to use we choose MySQL.

http://www.mysql.com/

4.5 GeoTools:

Geo Tools is an open source (LGPL) Java code library which provides standards compliant methods for the manipulation of geospatial data, for example to implement Geographic Information Systems (GIS). The Geo Tools library implements Open Geospatial Consortium (OGC) specifications as they are developed, in close collaboration with the GeoAPI project (an OGC working group). The reasons of why we choose GeoTools instead of GRASS or OpenGIS can be found in the first part of report.

http://geotools.codehaus.org/

http://geoapi.sourceforge.net/

4.6 Java APIs:

For almost all tasks we need to use Java APIs. For image process tasks we will use Java Image I/O API and Java Advanced Imaging API (JAI). GeoTools requires usage of these APIs.

Also for weather data gathering from web requires use of JAXP (Java API for XML Processing). Java 3D API (J3D) and Java OpenGL API (JOGL-optional) are other APIs that we will use second development term.

http://java.sun.com/javase/6/docs/

5. Diagrams

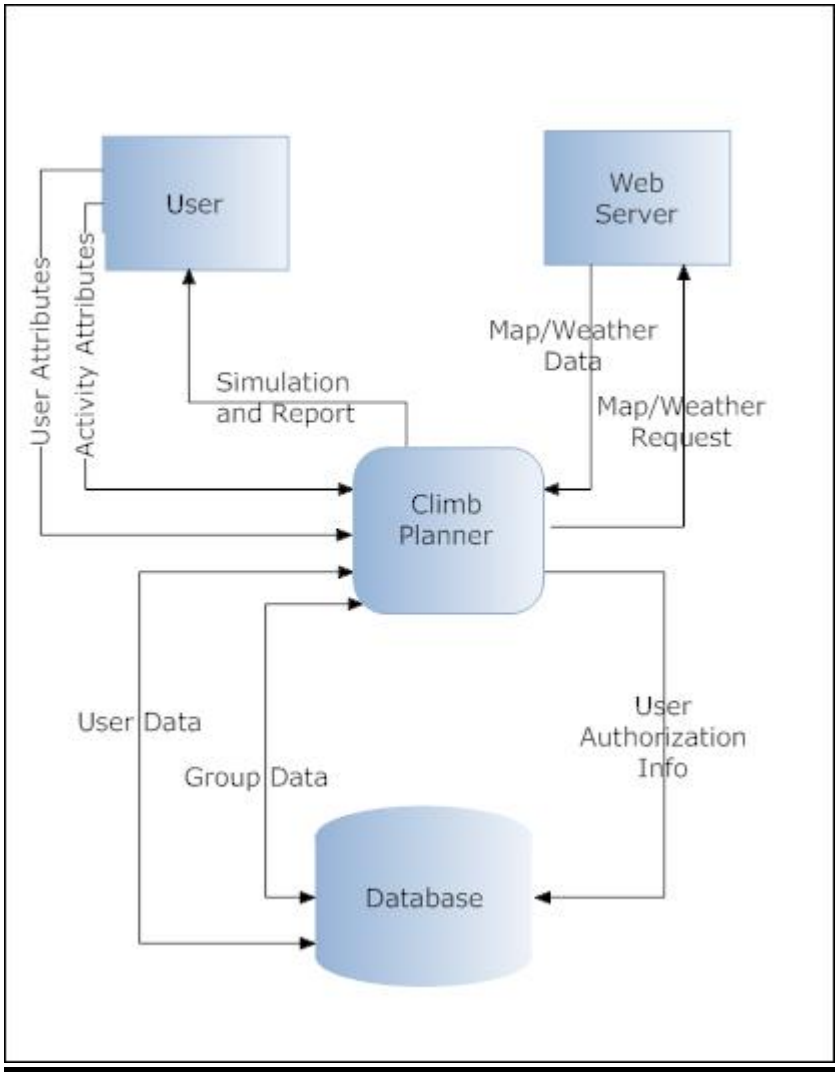


Figure 12: Level0 Data Flow Diagram for ClimbPlanner

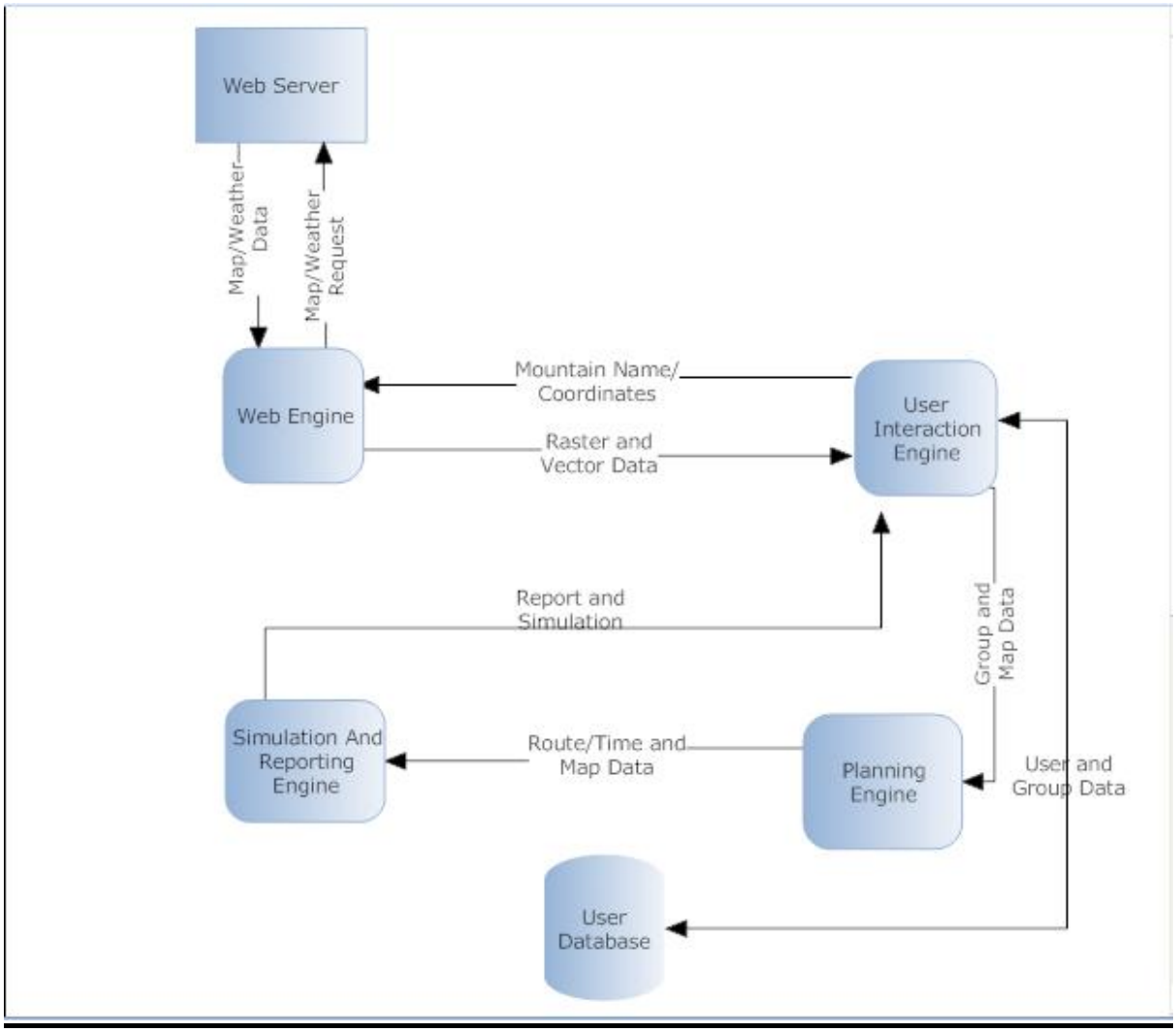


Figure 13: Level1 Data Flow Diagram for ClimbPlanner

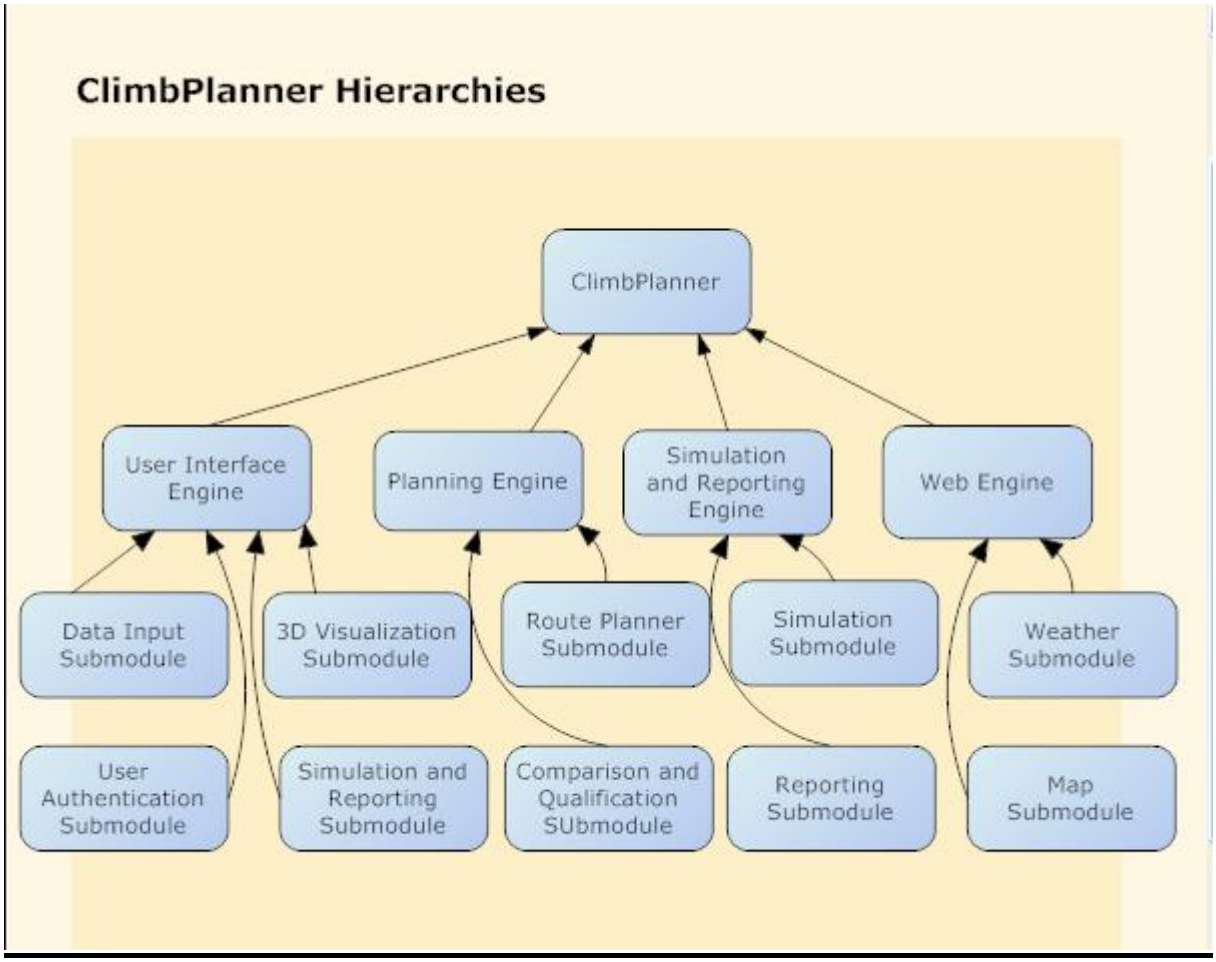


Figure 14: Hierarchical Diagram for ClimbPlanner

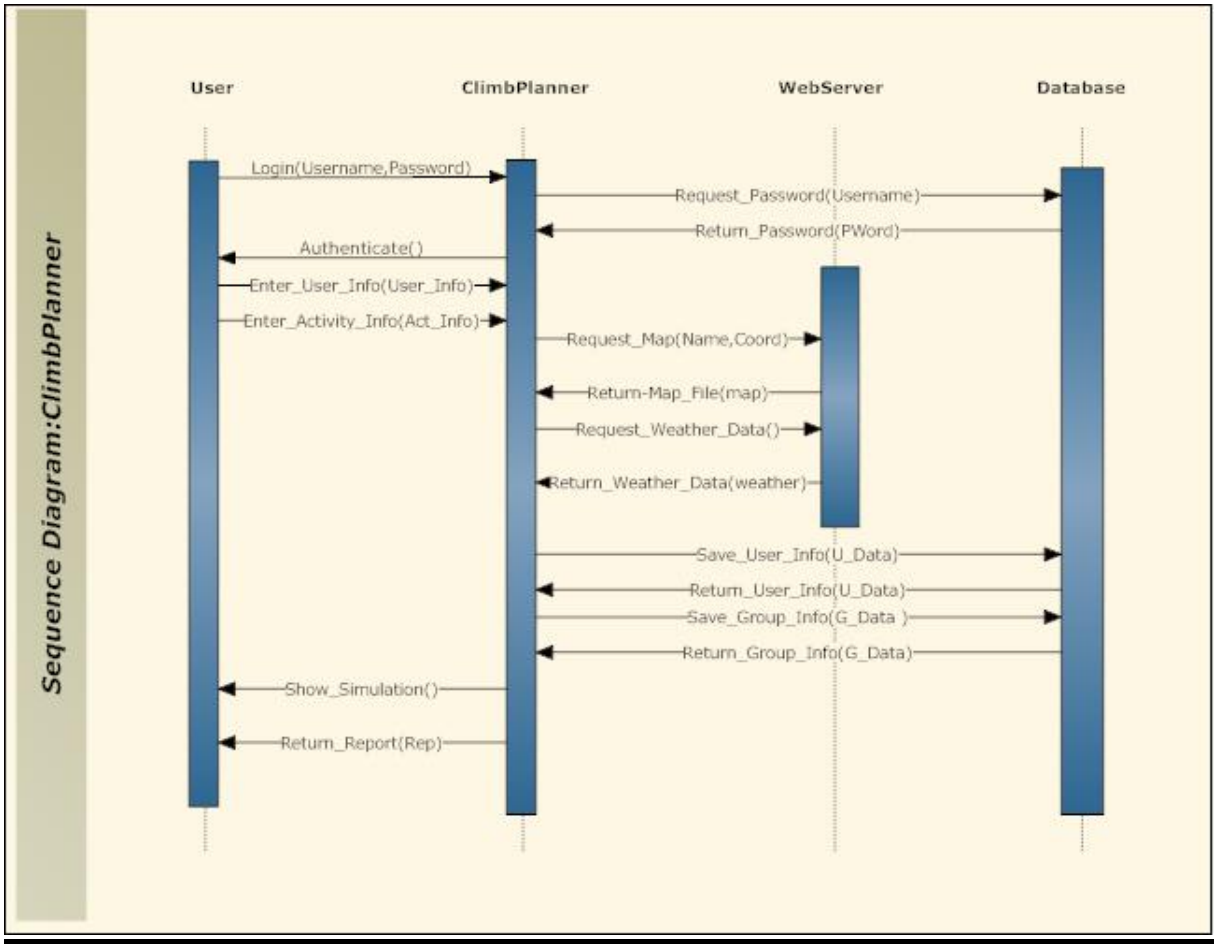


Figure 15: Sequence Diagram for ClimbPlanner

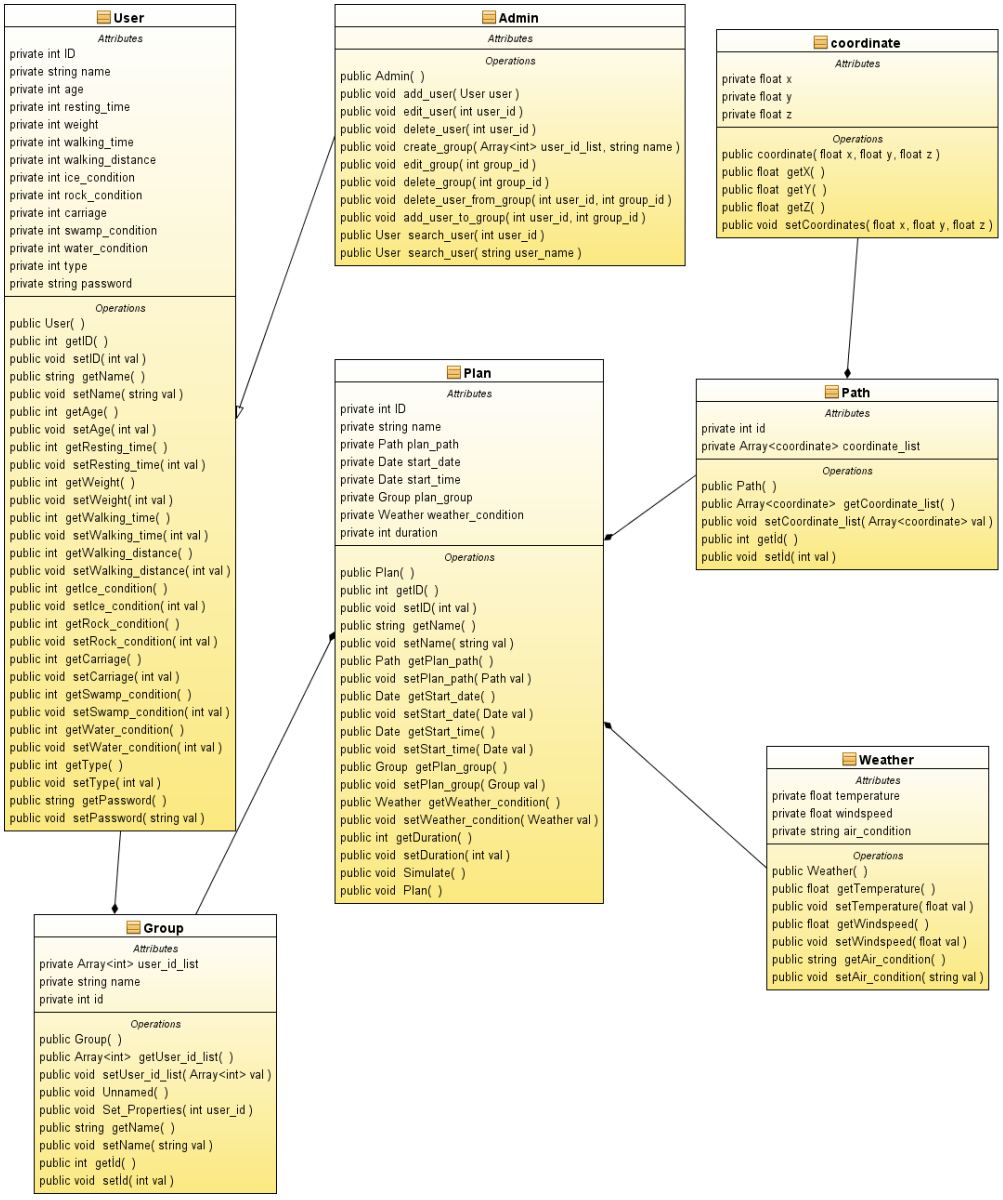


Figure 16: Class Diagram for ClimbPlanner

ClimbPlanner by Quattro Group

Page 1 of 2 19.1.09

Duration	Start Date	End Date	TASK
17d	**29.10.08**	**14.11.08**	**Analysis Report**
4d	29.10.08	1.11.08	Template for Analysis Report
4d	2.11.08	5.11.08	Research About GIS
3d	6.11.08	8.11.08	Research About Climbing
6d	9.11.08	14.11.08	Last Version of Requirements
54d	**24.11.08**	**16.1.09**	**Design**
12d	**24.11.08**	**5.12.08**	**Initial Design Report**
2d	15.11.08	16.11.08	ER Diagrams
6d	17.11.08	22.11.08	Initial User Interface
1d	23.11.08	23.11.08	Project Schedule
4d	23.11.08	26.11.08	Initial Component Level Design
4d	26.11.08	29.11.08	Defining Constraints
7d	29.11.08	5.12.08	Complete Report
40d	**8.12.08**	**16.1.09**	**Detailed Design Report**
11d	8.12.08	18.12.08	Final User Interfaces
11d	19.12.08	29.12.08	Detailed Component Level
18d	30.12.08	16.1.09	Forming Complete Report
74d	**14.11.08**	**26.1.09**	**Demo Preperation**
17d	14.11.08	30.11.08	Example User Interface
25d	27.11.08	21.12.08	Implementation Using
17d	17.12.08	2.1.09	Testing Modules
26d	1.1.09	26.1.09	Demo Integration
40d	8.12.08	16.1.09	Preparing Presentations

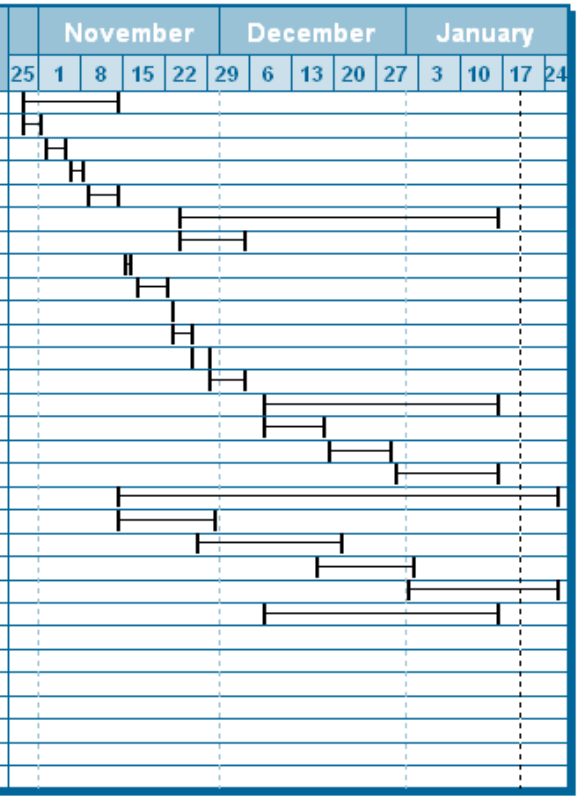


Figure 17: Gantt Chart (1st Term Development Process)

ClimbPlanner by Quattro Group

Page 2 of 2 19.1.09

Duration	Start Date	End Date	TASK
67d	**18.2.09**	**25.4.09**	**Engine Implementations**
5d	18.2.09	22.2.09	User Interaction Engine
13d	23.2.09	7.3.09	Web Engine
29d	8.3.09	5.4.09	Planning Engine
20d	6.4.09	25.4.09	Simulation&Reporting Engine
70d	**20.2.09**	**30.4.09**	**Testing of Modules**
4d	22.2.09	25.2.09	UIE Tests
17d	26.2.09	14.3.09	Web Engine Tests
26d	15.3.09	9.4.09	Planning Engine Tests
21d	10.4.09	30.4.09	Simulation&Reporting Engine
30d	**1.5.09**	**30.5.09**	**Final Tests**
11d	1.5.09	11.5.09	Integration Tests
7d	12.5.09	18.5.09	Alfa Tests
2d	19.5.09	20.5.09	First Release
5d	21.5.09	25.5.09	Beta Tests
5d	26.5.09	30.5.09	Improvement
19d	**26.5.09**	**13.6.09**	**Finalizing Project**
5d	26.5.09	30.5.09	Documentation
5d	31.5.09	4.6.09	Final Release
5d	5.6.09	9.6.09	Final Presentation Preperation
4d	10.6.09	13.6.09	Demo Preperation

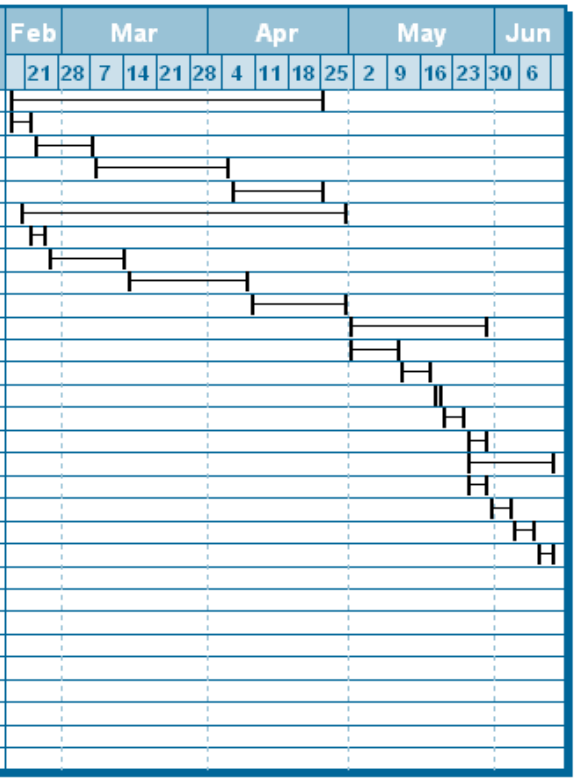


Figure 18: Gantt Chart (2nd Term Development Process)