Bachelor project TI
TI – ST

Version: 2.4
Status: Development
Date: Tuesday, 29 September 2009

Authors:
Jeremy Raes: 1199404
Menno Valkema: 1178326
Niels Brouwers: 1014838
Preface

The following report summarizes what has been an exciting and challenging assignment at Osaka University where we had the honour of being a guest for over sixteen weeks at the CyberMedia Center. A prestigious university, which embraces an international vision (“Live locally, grow globally”), matching our own ambition. This as part of our TU Delft BSc-program.

Our stay in Japan was of great personal and academic value as we experienced the daily life in one of the world leading countries; well known for its high-tech research and excellent schooling system. Recognized for its refined culture, which influenced the whole of Asia and the rest of the world.

Being one of the major cities in Japan, located in the Center of the Osaka-Kobe-Kyoto metropolitan area, Osaka has been the perfect host city for a better understanding of the Japanese way of life. The Japanese influence on modern day arts, innovating consumer products design, electronics and robotics. There is also much to learn as Japan is turning today’s problems – e.g. an aging population, a high population density, the rise of China and India in a world with ever decreasing resources, etc – into opportunities based on technology and smart engineering, Osaka and its university being part of this dynamic.

We would like to thank some people for their much appreciated effort, assistance and interest to our Bachelor project: Prof. S. Shimojo for assigning us. Dr. Y. Teranishi for his patience while helping and supporting in our work on his project and Dr. Ir. B. Sodoyer for reviewing our project goals and planning. J. de Vries (EWI - TUD), G. de Graaf (SFC - TUD) and S. Kondo (IC Hall – Osaka University) who helped us getting introduced at “Handai”. Our fellow students at the laboratory for making us feel right at home.

Jeremy Raes
Menno Valkema
Niels Brouwers
# Table of Contents

Preface ...................................................................................................................... 2

Introduction .............................................................................................................. 5
  Delft University of Technology, Netherlands ......................................................... 5
  Osaka University, Japan ......................................................................................... 5
  Bachelor project ..................................................................................................... 6
  DIPLOID .................................................................................................................. 9
  GMapWiki project ................................................................................................. 7
  Project leaders and team members ....................................................................... 7

Problem description and Analysis ...................................................................... 9
  Initial situation ..................................................................................................... 9
  Ideal situation ..................................................................................................... 10
  Realistic situation ............................................................................................... 10
  Challenges .......................................................................................................... 11
    Technical ........................................................................................................... 11
    Logical ............................................................................................................... 11
    Cultural .............................................................................................................. 12
    Communication and interaction ....................................................................... 13

Design .................................................................................................................... 13

Implementation ..................................................................................................... 14
  Changes ............................................................................................................... 14
  Modules ............................................................................................................... 14
    Sensors ............................................................................................................. 14
      MasterSensor ................................................................................................ 15
      ThinSensor .................................................................................................... 15
    Functionality .................................................................................................... 16
      Web ................................................................................................................ 16
      Personal ......................................................................................................... 17
      Other ............................................................................................................. 17
  Language ............................................................................................................ 18
  Libraries ............................................................................................................. 18

Evaluation ............................................................................................................... 18

Results .................................................................................................................... 19
  Implemented functionality .................................................................................. 19
  Extra functionality and requirements ............................................................... 20
  Experiences ......................................................................................................... 20

Recommendations ............................................................................................... 21

Literature ............................................................................................................... 22

Appendices ............................................................................................................ 23
  A. RAD document ............................................................................................... 23
    Requirements .................................................................................................... 23
      Functional Requirements .............................................................................. 23
      Non Functional Requirements .................................................................... 24
      Moscow Diagram ......................................................................................... 24
    Problem analysis ............................................................................................. 25
Introduction

This chapter describes the different institutions, the bachelor project itself, the project leaders and the team members.

**Delft University of Technology, Netherlands**

Delft University of Technology (TU Delft) was founded in 1842 by the Dutch King, William II. It is the oldest and largest University of Technology in the Netherlands. The 7 faculties offer 17 bachelor’s courses and 29 Master’s courses in comprehensive programmes of education and research.


The members of this research project are all three registered as bachelor students in computer science.

**Osaka University, Japan**

Osaka University founded in 1869 as “Osaka prefecture medical school”, is the 6th oldest university in Japan and considered as one of the most prestigious academies in the country and Asia. After a merger in 1949 between “Naniwa Higher School” and “Osaka Higher School”, the university grew from five to ten different faculties: Letters, Human Sciences, Law, Economics, Science, Medicine, Dentistry, Pharmaceutical Sciences, Engineering and the school of Engineering Science.

The CyberMedia Center (CMC) was founded in April 2000 as Osaka University sought to reorganize and expand its Computation Center (CC) to form its branch of the Information Infrastructure Center. In the expansion, Education Center for Information Processing and a part of the university library were merged into CMC. While CC continues to provide computers for the utilization of advanced scientific techniques and media services, the Education Center for Information Processing contributes by promoting education in information processing and the university library by providing digital contents.¹

¹ CMC-text by CyberMedia Center
**Bachelor project**

Students enrolled in the third and last year of the Delft University of Technology (TU Delft) Computer Science (part of a five year master in engineering), have to complete their undergraduate programme with a practical (group) assignment in Software Technology.

This Bachelor Project is a 9-week (or more) long project and performed in groups of two to five students working together as a team on a single assignment. The project has to be related to the computer sciences, using techniques learnt in the bachelor programme. This gives a broad variety of possibilities in either existing or new projects:

- Problem analysis
- Design specification
- Prototyping
- Code implementation
- Code re-factoring

**DIPLOID**

This section will give a quick introduction the DIPLOID system, the project team developed at the Cyber Media Center. The introduction gives the reader basic knowledge about the system for understanding some other sections in the system.

The acronym DIPLOID stands for DIstributed P2P LOcation-based Information Database. The development of DIPLOID is the main assignment of this BSc project. The purpose of this information storage system is to store location-based GMapWiki-information in a flexible extensible way using PIAX, a distributed storage system developed at Osaka University. To fulfill these goals, information is stored as a peer-to-peer system and the information is distributed over the peers in the system.

In short, DIPLOID is a two way interface between a front-end application and a storage layer.

Using DIPLOID, users should be able to create their own custom sensors, to add information to the location-based-wiki. The information added is stored on the sensor itself but it is indexed by the peer-to-peer system, to make the data accessible by all users in the system. In the system are several different sensors:

- HeavyPeers is used to store peer-to-peer indexes and the information published by the sensor itself.
• LightPeers are used to connect unstable clients with the platform. They can retrieve or publish information on the network; however a LightPeer does not store any data itself.

The DIPLOID system will be used as a storage and search system for the GMapWiki system and will replace the relational-storage system in the future.

**PIAX**

PIAX is a peer-to-peer platform on top of which the DIPLOID project is being developed. It is written in Java, and offers it’s users a portable, flexible, scalable and powerful agent-based framework. It uses LLNET, a technique similar to a Distributed Hash Table using geographically linked keys to perform CRUD- (Create, Read, Update, Delete) operations on its nodes.

The PIAX technology is developed by the CyberMedia Center at Osaka University.

**GMapWiki project**

GMapWiki (http://gmapwiki.org) is an online front-end which uses the DIPLOID layer for data storage.

Peer-to-peer systems are becoming an increasingly popular means for distributing computation and storage resources because of their robust, dynamic and scalable nature. Goal of this project is to present a prototype that allows users to create, modify and share markers on GoogleMaps as Wiki content. Key design requirements were distribution of both network load and storage, locality of private information and code maintainability.

**Project leaders and team members**

Project leaders:

_Shinji Shimojo_ received the ME and a DE degrees from Osaka University in 1983 and 1986, respectively. From 1986 to 1989 he was an Assistant Professor in the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. Since 1998 he was an Professor in CyberMedia Center, Osaka University. He was engaged in the project of object oriented multimedia presentation system called Harmony. His current interests cover wide diversity of networked applications such multimedia database, grid computing, networked virtual reality and so on. He is a member of ACM and IEEE.

Prof. Shimojo is director of the Osaka University CyberMedia Center.
Yuuichi Teranishi received his ME and PhD degree in Information and Computer Sciences from the Osaka University in 1995 and 2004. He is currently an Associate Professor of Applied Information Systems Laboratory of CyberMedia Center, Osaka University, Japan. He was an employee of NTT, Japan from 1995 to 2004. His current research interests include the availability and security of ubiquitous content distribution systems, especially on the peer-to-peer architecture. He is a member of the Information Processing Society of Japan and an Organizer of the Special Interest Group on Distributed Processing System (SIG-DPS).²

As head of the Ubiquitous Computing research group at CyberMedia Center, Dr. Teranishi is the head supervisor to the GMapWiki-project.

Dr. Bernard Sodoyer is a teacher at Delft University of Technology and the main coordinator to the Computer Science Bachelor Project.

Team members:

In alphabetical order:

Jeremy Raes graduated in 2002 from high school and started his studies in Computer Science at the College of Antwerp. He is currently an undergraduate student at Delft University of Technology in Netherlands (third year of a five year master programme in engineering) and a guest student at the CyberMedia Center, Osaka University from April 2006 until August.

Specific role in the project: overall progress and communication.

Menno Valkema is enrolled as a guest student in Computer Science at the CyberMedia Center, Osaka University from April until August. After graduating from High School in 1999 he first joined the College of the Hague (Holland) and continued his studies at Delft University of Technology.

Specific role in the project: design.

Niels Brouwers started working as a programmer for GameBoy advance after his high school graduation in 1999; Currently in his third year as an undergraduate at Delft University of Technology and guest student at the CyberMedia Center from April 06 until August 06.

Specific role in the project: PIAX-platform.

² Curriculum Vitae S. Shimojo and Y. Teranishi are provided by CyberMedia Center
Problem description and Analysis

Initial system

This section describes the initial status of the DIPLOID project; the limitations; how the initial system can be improved; the special interest groups are identified and a summarized overview of the root of the problems in the initial situation will be given.

Description of the initial system:

Entries and user information in (the initial) DIPLOID are stored in a relational database. This data is read and modified through the web interface: GMapWiki. When a map is requested this happens: The entries in a given area are retrieved from the database. These entries will be filtered given some constraints. The images for the GMapWiki front-end are retrieved form GoogleMaps. Finally the entries and the images are merged and displayed at the user’s computer. Entries can only be added to the DIPLOID layer trough the GMapWiki web interface.

Problems with the initial system:

The initial system stores all the user information and entries in a relation-based centralized database. Such systems are limited in scalability (the system can only handle a limited number of requests) storage capabilities and are very vulnerable against possible malfunctions (e.g.: hard disk failures) or unforeseen accidents (e.g.: fire). The absence of an accessible interface makes it in the initial system nearly impossible to develop a series of different front-ends.

Testing showed that the initial DIPLOID implementation proofed to be too time-consuming for any practical usage.

The new system should provide a more flexible system to add, remove, retrieve and modify entries so other front-ends can easily contribute/modify data to the network. As the data and the usage of the GMapWiki system grows on a daily bases, a more scalable data store is required to support future growth and speed.

Interest groups

Three different interest groups can be identified:

- Maintainers administrate the different peers that make up the DIPLOID system.
- Viewers are individuals who use a front-end (e.g. GMapWiki) to enter, view, modify, and remove data to the DIPLOID system.
Automated agents are just like viewers but contribute information to the network without any human interaction.

**Ideal system**

This section describes how the ideal situation would look like. This may not be feasible but gives a goal to strive for during the design and development.

In the perfect system there are two things that should be improved: data access and data storage. It's easy and fast to add, modify and delete notes. The entries are easy available through programming interfaces, so it will be easy to add automated sensors to manage entries. The data is stored in a scalable way, so it must be easy to add more storage to the system. Storage hosts must be able to join the system freely, and the system itself handles the distribution of the data. Whenever a part of the system goes down, it must be possible to recover automatically, without the user noticing it.

**Realistic system**

The ideal situation may not be realistic, because of technical-, financial- and other limitations. Therefore in this section, a more realistic situation will be given, creating more realistic project goals.

The hardest part of the ideal situation is automatically recovering from host failures. If each host is equal and hosts can join freely, it is hard to assert each host to be of high quality being able to recover from system crashes. Even though it's impossible to predict which system and how many systems will fail. Therefore the peer-to-peer system can't guarantee the data stored on one or more systems will always be available and recoverable.

To overcome these problems there should be only high quality hosts in the peer-to-peer network responsible for storing data. These hosts should be able to recover from their own failures and never leave the peer-to-peer network. All data is indexed in a LLNET layer with references to database entries. Other peer-to-peer indexes can be added if needed, for example when searching entries by author. In this way the system is able to recover from failures, database systems can be added on the fly.

It's impossible for a single peer to get the global picture of the entire system because of the peer-to-peer based architecture. Peers only know their own neighbours and can retrieve some information based on the indexes available. Therefore a load balancing scheme can only be implemented if there are 'all knowing hosts' available in the system, which keeps track of all storage based peers.

The data will be available through a programming interface available for sensors and web interface applications. These 'users' of the storage system, won't be responsible for storage of data: only for modifying and requesting data.
Challenges

The challenges section describes which challenges there are from the team's point of view. The project challenges are viewed from three directions: technical, logical and cultural.

Technical

The first technical challenge is probably the use of peer-to-peer technologies. This is a challenge for the team because none of the team members has previous experience in developing peer-to-peer systems. The overcome this challenge the team will study papers on peer-to-peer systems in the first few weeks of the analysis phase of the project.

Related to this topic is the peer-to-peer library the team will use. This library implements many of the technologies discussed in the studied papers. The library used is fully documented in Japanese and still subject to development. Some parts are more stable than other parts of the library. The project team has to work close with English speaking Japanese laboratory partners. The PIAX system will be isolated from the other parts of the system to prevent changes made in the PIAX system effect the teams' work. Therefore a Facade Pattern will be used in the design in order to keep the link between both layers to a strict minimum.

During the development process multiple team members will work on the same parts of code. To prevent conflicts in the work done by the team members, the team will use a version control system, which manages all code in the project. To check if the code compiles and works well, the system will check on a daily basis if the code compiles and all test cases succeed.

Much of the code and documentation the team will use is written and documented in Japanese. To understand this data the team will use translation tools and help of Japanese lab partners. The current storage system used is based on a relational database. This storage system provides many complex data access methods. Building a distributed system with equal functionality is a complex task. The project team will focus on implementing only the functionality used by the GMapWiki system.

Logical

Distributed networks in which each peer is free to join or leave, come with a number of challenges. Although not all of these challenges are within the scope of the assignment, it’s interesting noting them.

• Churn Resilience:
  This problem arises with the continuous node arrival and departure in a Distributed Hash Table (DHT). Three factors are affected under churn: reactive versus periodic failure recovery, message timeout calculation, and proximity
neighbour selection.

Churn Resilience occurs in the PIAX layer. Therefore it is not possible to address this problem from the Deploid layer. However: the project will have to distinguish between classes of peers: HeavyPeer for stable peers, responsible for data storage, and a LightPeer for unstable peers, which cannot store data itself but delegate this task.

- Security and private data storage:
  A SOAP-interface is used to send data between the different peers. Although this data might be private, the prototype will not have any encryption possibilities. The platform can store personal data locally.

- PIAX still in development:
  As the used P2P platform is still in development, functionality might still change, re-factoring the code time consuming. DIPLOID provides a Façade-class to reduce dependencies of outside code on the inner workings, responsible for handling all PIAX-functionality.

- SOAP-calls:
  Every time a user scrolls the screen, the system will download all the visible entries again and thus putting extra load on the network. This problem can be solved adding a caching mechanism.

**Cultural**

Prior to our sojourn in Japan, we participated on advice of the Delft Student Facility Centre (SFC) in a workshop discussing intercultural communication. Table 1 plots Power Distance, Individualism, Masculinity, Uncertainty Avoidance and Long Term Orientation rate of a culture. Although the stated properties are a generalization and do not necessarily hold for a (more relaxed) university environment, they sketch the differences that exists between countries.

<table>
<thead>
<tr>
<th></th>
<th>PDI</th>
<th>IDV</th>
<th>MAS</th>
<th>UIA</th>
<th>LTO*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Japan</td>
<td>54</td>
<td>46</td>
<td>95</td>
<td>92</td>
<td>80</td>
</tr>
<tr>
<td>Netherlands</td>
<td>38</td>
<td>80</td>
<td>14</td>
<td>53</td>
<td>44</td>
</tr>
<tr>
<td>Belgium</td>
<td>65</td>
<td>75</td>
<td>54</td>
<td>94</td>
<td>38</td>
</tr>
<tr>
<td>MIN</td>
<td>13 (ISR)</td>
<td>8 (ECA)</td>
<td>5 (SWE)</td>
<td>8 (SIN)</td>
<td>0 (PAK)</td>
</tr>
<tr>
<td>MAX</td>
<td>104 (MAL/SLK)</td>
<td>90 (AUL)</td>
<td>110 (SLK)</td>
<td>112 (GRE)</td>
<td>118 (CHN)</td>
</tr>
</tbody>
</table>

ISR = Israel; MAL = Malaysia; SLK = Slovakia; ECA = Ecuador; AUL = Australia; SWE = Sweden; SIN = Singapore; GRE = Greece; PAK = Pakistan; CHN = China
*Not all countries are rated.

**PDI – Power Distance**

Low
- Low dependence needs
- Inequality minimized
- Hierarchy for convenience
- Superiors accessible
- All should have equal rights

High
- High dependence
- Inequality accepted
- Hierarchy needed
- Superiors often inaccessible
- Power holders have privileges

**IDV – Individualism**
Communication and interaction

Code, tests and documents are shared with group members via an SVN-server. Supervisors have local access to this repository via a web interface for follow up. Problems, questions, changes and updated within the group are handled in an unconventional way by short planned or unplanned/improvised meetings whenever requested by a group member. Questions concerning requirements and design towards Teranishi-san are communicated using mail or short meetings.

During weekly UP2U-meetings progress is communicated towards the other members of the Ubiquitous Networking-research laboratory; focused on encountered problems. This way all members can reflect on them, using their own expertise to try finding an answer to the given problem. Dr. Ir. Sodoyer is kept up to date by mailing weekly progress reports. In these reports the project-team will consider the progress of the GMapWiki-project and an updated planning.

Each team member has an Osaka University assigned student-assistant, helping out with project or university related difficulties.

Design
The design chapter will give a brief overview of the design process covering: challenges, improvements.

Architectural Design

The architectural design describes how responsibilities will be divided over different systems. The Architectural Design will be checked carefully with the supervisor, because of the limited experience of the project team with peer-to-peer based architectures. After the project team completes the Architectural Design and satisfies the requirements of the university the object oriented design can be made. For the Architectural Design it was important to give all peers equal functionality but different responsibilities. There had to be a difference between heavy peers and light peers, in sense of responsibilities. For more details on the solution see Appendix A.

The object oriented design describes an object model where is defined how responsibilities are divided between objects in the software. The requirements for the object-oriented design was designing an application library flexible enough to support the requirements developed in the RAD document (See Appendix A). The library should be useful for 3rd parties developing their own GMapWiki sensors. The design must be easy to understand for programmers and implement. But the design can't be too simple because then it will be hard to extend the design in the future. Finally during the design should automated testing kept in mind. (See Appendix A for the object oriented class diagram).

Implementation

Changes

The new storage system will eventually replace some parts of the current system. This implies that in the ideal system the interface of the new system is the as the old system so the old system parts don't need to be touched.

While the interface of the current storage implementation has a quite complex interface (DBI and sql), It saves trouble to change some of the current system parts to work with the new storage system. For this we change the presentation interface of the system. In the ruby presentation code all storage functionality is encapsulated in a single class, so the only change in the current system is in the data access class.

Modules

Several modules need to be developed to realize the system. By distributing the system functionality over multiple modules the per-module-complexity goes down which decreases development time and the number of bugs. When functionality is decoupled and put in multiple modules, it is more easy to distribute the computation operations over multiple systems and limits the per system computation power needed.

In this section the modules to be developed during the project will be listed and the functionality they provide.

First a list will be provided with the functionality a module can provide, each module will provide one ore more of the operations listed in the table below.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntryRegistration</td>
<td>A module providing EntryRegistration services, provides methods for adding and removing EntryReference's to the WorldMap managed by piax.</td>
</tr>
<tr>
<td>EntrySharing</td>
<td>A module providing EntrySharing services provides methods for requesting it's local data store for entries.</td>
</tr>
<tr>
<td>WorldMapSearch</td>
<td>A module providing WorldMapSearch offers functionality for searching the worldmap for entries to be displayed.</td>
</tr>
<tr>
<td>UserSearch</td>
<td>Modules providing UserSearch can be used to locate buddy lists stored in the system.</td>
</tr>
</tbody>
</table>

**Sensors**

A sensor uses the peer-to-peer system and is able to publish its data. The system will start out with two kinds of sensors: MasterSensors and ThinSensors. A sensor can have several kinds of functionalities (see next section).

**MasterSensor**

A master sensor is a strong stable peer, which offers much computation- and storage-power. Therefore the host is strong enough to keep up the peer-to-peer network and provide services to hosts that are not really strong. The MasterSensor provides the following functionality for external hosts: EntryRegistration, EntrySharing, WorldmapSearch and UserSearch. EntrySharing is used for publishing it's own added entries, all other services are provided for letting less strong hosts (ThinSensors) use the system.

**ThinSensor**

A ThinSensor is a computer that can't guarantee availability. Examples of this kind of hosts are computers using a telephone-line connecting to the Internet, PDA-users and Cell phone users. ThinSensors should be able to share information on the wiki-storage, but shouldn't be responsible for keeping up the peer-to-peer system. Therefore a ThinSensor uses a MasterSensor for sharing its references. When a reference to a note is requested, the requesting host connects to the ThinSensor and downloads the reference that should be found.
Functionality

Web

The web server will need access to the WorldMap class for retrieving references. This sensor should also be able to add entries and store them.

There is already an existing demo of a web interface implementation (by Dr. Y. Teranishi) connected to a non-distributed implementation, using the Ruby scripting language. As this version already offers all the required functionality, it was only necessary to update the Ruby-code in order to connect the front-end with the back-end. This was agreed upon after consulting Teranishi-san.

The web interface allows users to:

- Add, view, edit and remove entries
- Log on / Log off (http://www.sixapart.com/typekey/)
- Use the Buddy List functionality
Personal

A personal sensor will provide functionality to share your personal bar, current location, home location, etc.

Other

It shouldn't be hard to add new functionality to sensors or to create new sensors.
Language

This section gives an overview of the used languages and why they are used.

Java:
Java is the language of choice in this project to implement DIPLOID, the new peer-to-peer data store, and replace (partly) the relational database. Java is a widely used object oriented programming language praised for its multi platform support and the loads of available classes in its standard libraries.

For the GMapWiki project Java is the programming language of choice. The reasons are that it's preferable to run the peer-2-peer application on multiple platforms and a peer-to-peer library is used (PIAX) supports only Java.

Java script:
Java script is used only in the browser to interface with GoogleMaps to add notes to the maps. Since there is no need to modify this already functional code, Java script won't be discussed in this report.

Ruby:
Ruby is a fully object-oriented interpreted programming language. It supports many libraries and is designed to create programs in little time. Ruby is used for the web interface implementation. It generates the XML documents containing the entries that should be displayed. For this project a small part of the Ruby code should be modified: the part that retrieves the data from the database.

Libraries

For developing the system several libraries are used. XSoap, PIAX, JUnit and Aglets. XSoap is a library to export the functionality provided by a class as a SOAP webservice. XSoap is used because it can export classes fully transparent to the programmer. The PIAX library provides the peer-to-peer functionality for the system: for this project the DHT and LLNet functionality is used. This library is developed at the CyberMedia Center in Osaka. The PIAX library depends on the Aglets library. Aglets provides functionality to move agents from one peer to another peer fully transparent to the programmer.

Finally the JUnit library is used. This library doesn’t provide any functionality to the system. JUnit is only used for unit-testing the system.

Evaluation

The evaluation section will aim for defining the constraints on which the project is evaluated. Goal of the GMapWiki Storage project is deliver a working prototype for the P2P distribution of geographically linked data. The project will be evaluated by the following constraints:

Quality of the code:
- The code should be self explanatory.
- If possible, every class should be tested. By preference using tools as JUnit, ea.
- The code should be easy to re-factor and maintain, using, where possible, well know design
patterns
• Design by contract should be enforced by using assertions.
• Missing libraries should be simulated with place holder code.

Requirements:
From the MoSCoW diagram (see Appendix A):
• All the ‘Must haves’ should be implemented.
• ‘Should haves’ should be implemented. If not, they should be well documented in the recommendation section.
• ‘Could haves’ that aren’t implemented should be listed in the recommendation section and documented if necessary.
• ‘Would haves’ that aren’t implemented should be listed in the recommendation section.

Documentation:
• The code will be documented using Java doc
• Every document should be self sufficient

Results

Implemented functionality

<table>
<thead>
<tr>
<th>Requirement (No)</th>
<th>Status (07/08/2006)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed (1)</td>
<td>Implemented (Must)</td>
</tr>
<tr>
<td>Entry note storage (2)</td>
<td>Implemented (Must)</td>
</tr>
<tr>
<td>Cashing entries (2.1)</td>
<td>Implemented (Could)</td>
</tr>
<tr>
<td>User storage (3)</td>
<td>Implemented (Must)</td>
</tr>
<tr>
<td>Viewing notes (4)</td>
<td>Implemented (Must)</td>
</tr>
<tr>
<td>Adding notes (5)</td>
<td>Implemented (Must)</td>
</tr>
<tr>
<td>Removing notes (6)</td>
<td>Implemented (Must)</td>
</tr>
<tr>
<td>Local entry management (7)</td>
<td>Implemented (Should)</td>
</tr>
<tr>
<td>Easy programmable entry filtering interface(8)</td>
<td>Implemented (Should)</td>
</tr>
<tr>
<td>Filtering by buddy (8.1)</td>
<td>Implemented (Could)</td>
</tr>
<tr>
<td>Filtering by region (8.2)</td>
<td>Implemented (Could)</td>
</tr>
<tr>
<td>Filtering by type (8.3)</td>
<td>Implemented (Could) See design*</td>
</tr>
<tr>
<td>Other types of filtering (8.4)</td>
<td>Implemented (Would) See design*</td>
</tr>
<tr>
<td>Range query support (10)</td>
<td>Implemented (Must)</td>
</tr>
</tbody>
</table>
### Extra functionality and requirements

This chapter summarizes the additional functionality, added when the requirements where already agreed upon (see also RAD document: Requirements). These extra functions were implemented for a variety of reasons.

- A small proof-of-concept, SWT-based, stand-alone client was written to test the system manually while the ruby implementation was still under development. This was not developed further as it was not considered part of the project, and as such was not merged into the official code base, nor was it included in the final release.

- Because of the lazy-loading scheme employed when retrieving entry objects through SOAP, every object is loaded individually. In addition, entry objects may be loaded multiple times even though their content does not change. To reduce communication overhead, a caching mechanism has been added. When entry objects are retrieved from remote locations over SOAP, they are held in memory for a certain amount of time. After this ‘timeout’ has expired, they are discarded and the system will try to reload them. Parties that add entry references to the system can set the timeout property freely. This allows both the disabling of the caching mechanism by setting a timeout of 0 milliseconds, as well as forcing the caching system to ‘never’ discard the cached copy by setting an extremely large timeout value.

- Reference entries may point to a remote machine that may be unreachable due to technical problems. A block-list system has been implemented that will ensure that remote machines will not be continuously retried once a communications-error has occurred. After a certain time the host will be removed from the list so that a remote host will not be blocked indefinitely.

### Experiences

Our BSc project was a unique opportunity as we had the chance of cooperating in an interesting and promising research topic at Osaka University, Japan.

All three of us had already some experience with distributed systems, larger projects in various companies and have been working together in previous computer science projects (Menno and Niels:
Distributed, Computer Graphics; Jeremy and Niels: Fundamental Computer Science practicum; Jeremy and Menno: Embedded systems). Nonetheless; this assignment proofed to be challenging and life changing: we learnt to work better and closer as a team plus got an insight into the Japanese culture, way of life and educational system.

The project started the project by studying P2P papers, concentrating on various techniques such as DHT, ZNET, SkipList, LLNET and load balancing schemes. We concluded this research period with a presentation for our fellow students, discussing what we’ve learnt.

Every team member got a set of responsibilities,

Japan is cool
Experience programming in java and working for companies on larger software projects
A broad interest in software/design related techniques and literature
Working in another country and different country is a life changing experience
Osaka University was very pleased with our skills

**Recommendations**

The current DIPLOID is build as a prototype. In this case it means the development team tried to write a well-designed and well-written piece of code containing basic functionality. This section will give an overview what functionality the project team thinks there should be added before the system is 'production ready'.

Currently security is left out of the core system. The topic came up during the design phase but the team decided (after discussing it with the tutor) to leave this out for the prototype. Reasons for this were that most security features needed belonged in the peer-to-peer library and the features we could implement were a bachelor project on their own to do it the way it should be done. The P2P is used to store references to entries and user information. References and user entries can be read and written by anyone having PIAX access. The entries and information that belongs to a user, should have more security levels.

Security is also missing in the SOAP interface. This interface currently doesn't check who or what requests data. In future there should be a form of authentication.

In a production environment it is required to have more control over this process. For example some user-details should only be available for the friends of a user. An entry may be removed only by its owner, but should be edited only by its friends. To implement these features the 'right' way, advanced knowledge of cryptography is needed and development-knowledge and access to the peer-to-peer layer of the system is needed.
The DBI storage interface, the system stored the history an entry. Our supervisor decided to leave it out and focus on the distribution and flexibility of the information in the system. In future there should be a way to request the history of an entry in the system.

Because a sensor stores its own entries, an entry will be lost when it leaves the system. This is done to let every user have responsibility for its own entries and make sure only information that really exists is stored in the system. In future it might be desired to store information in the system that stays even if the sensor leaves the network.

The data is currently indexed by location. This implies that searching a certain area for information is handled fast by the system. There are cases when indexing only by location isn't enough. If a user wants to find all restaurants in the system he might want to search all entries by category or by user. Therefore other p2p indexing methods should be added.

Automated purging of ‘dead’ references: When a host that serves entries goes down without notice, some references may remain in the system that point to this host. Clients retrieving those entries will keep trying to query the host even though it’s gone. The P2PFacade implementation should therefore occasionally purge all references to point to ‘dead’ hosts.

**Literature**

**Papers:**

- MapWiki: A Map-based Content Sharing System for Distributed Location-dependent Information; Yuuichi Teranishi et al.
- Kademlia: A Peer-to-peer Information System based on the XOR Metric; Petar Maymounkov and David Mazi`eres
- A Location-Based Peer-to-Peer Network for Context-Aware Services in a Ubiquitous Environment; Yu Kaneko et al.
- A Case Study in Building Layered DHT Applications; Yatin Chawathe et al.

**Books:**

- Design Patterns: Elements of Reusable Object-Oriented Software; Gang of four
- Refactoring: Improving the Design of Existing Code; Martin Fowler
- Distributed Systems: Principles and Paradigms; Andrew S. Tanenbaum, Maarten van Steen
Appendices

A. RAD document

This RAD document contains the requirements, analysis and design information for developing flexible distributed data storage for GMapWiki.

Requirements

This section contains the requirements for the system. The requirements will be listed by priority and a short description will be given per requirement. By assigning a priority to each requirement, its the system can be build step by step building, the high priority components first.

There are two kinds of requirements: Functional requirements, it describes which functions the system must have. Non functional requirements, this are all requirements which aren't function related. Finally all functional requirements will be prioritized in a Moscow diagram.

Functional Requirements

The functional requirements listed below define the internal workings of the software and the functions which the system should support, describing the (technical) behavior of the system.

1. Distributed; The storage and calculations should be distributed over multiple hosts.
2. Entry note storage; Each peer presenting notes should store it's own entry notes.
3. User storage; The database of users should be distributed over all hosts in the system.
4. Viewing notes through web interface. The presentation layer of the system should be modified to present the data in the system.
5. Adding notes through web interface. The presentation layer of the system should be modified to have functionality to add entry-notes.
6. Removing notes through web interface. The presentation layer of the system should be modified to have functionality to add and remove entries.
7. Managing notes locally. CRUD operations on own notes should be handled locally.
8. Easy entry filtering. It's not always necessary to display all notes all the time, therefore some filtering mechanism must be implemented. Forms of filtering can be:
   - By buddy
   - By region (is implemented as part of LL-NET)
   - By type (example: only restaurants)
   - Other…
9. It should be possible to select filters
10. Range queries for entry notes. When the presentation system displays entries on the screen, range queries should be supported.
11. Local Storage of entry notes. Every peer should be responsible for managing its own entry-notes. So when a peer goes off line its data is lost.
12. Easy to add new sensors. The system should be flexible enough to easily add special purpose sensors. Possible examples of such sensors are:
   - Personal Sensor: Sensor publishing personal notes of a person like: home location, current location, etc.
   - Weather sensor: Giving weather information for a certain area
   - Other...

**Non Functional Requirements**

The non-functional requirements section describes the non functional requirements of the system and specify criteria that can be used to judge the operation of a system, rather than specific behaviours.

1. A stable and flexible design has priority over code (prototype)
2. The storage system should be written in Java.
3. The presentation layer should be written in Ruby.
4. Users should be able to develop sensors by themselves.

**Moscow Diagram**

This MoSCoW diagram (Must have, Should have, Could have, Won’t have) functional requirements are prioritized according to importance. Functionality with lower priority will only be implemented after high priority functions.

<table>
<thead>
<tr>
<th>Description (No)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed (1)</td>
<td>Must have</td>
</tr>
<tr>
<td>Entry note storage (2)</td>
<td>Must have</td>
</tr>
<tr>
<td>Cashing entries (2.1)</td>
<td>Could have</td>
</tr>
<tr>
<td>User storage (3)</td>
<td>Must have</td>
</tr>
<tr>
<td>Viewing notes (4)</td>
<td>Must have</td>
</tr>
<tr>
<td>Adding notes (5)</td>
<td>Must have</td>
</tr>
<tr>
<td>Removing notes (6)</td>
<td>Must have</td>
</tr>
<tr>
<td>Local entry management (7)</td>
<td>Should have</td>
</tr>
<tr>
<td>Easy programmable entry filtering interface (8)</td>
<td>Should have</td>
</tr>
<tr>
<td>Filtering by buddy (8.1)</td>
<td>Could have</td>
</tr>
<tr>
<td>Filtering by region (8.2)</td>
<td>Could have</td>
</tr>
<tr>
<td>Filtering by type (8.3)</td>
<td>Could have</td>
</tr>
<tr>
<td>Other types of filtering (8.4)</td>
<td>Would have</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Description (No)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range query support (10)</td>
<td>Must have</td>
</tr>
<tr>
<td>Easy programmable sensor interface (12)</td>
<td>Should have</td>
</tr>
<tr>
<td>Personal Sensor (12.1)</td>
<td>Should have</td>
</tr>
</tbody>
</table>

**Problem analysis**

The problem analysis section describes the problem that should be solved. This is divided in three sections: first the current system.

**The current system**

GMapWiki is a service that allows its users to view, add and edit location-based information on a world map. The goal is to let users share information about places, and thus provide a comprehensive, open and free encyclopedia of the world around us.

The present system is built on top of the Google Maps API. This method uses AJAX technology inside a web browser to create an interactive map system without page reloads. The system can be easily augmented with vector data and overlay information in the form of *markers*. A marker is a point on the map that has some information attached to it. Examples of such markers could be a good curry restaurant, a nice park, or someone’s home.

GMapWiki aims to augment Google Maps with a system for sharing this location-based information. At present the system consists of two main modules. On the client-side AJAX code requests, processes and displays the markers. On the server-side, ruby code acts as an interface between the client and a relational database.

The current version is a proof of concept. It works well, but is not very scalable. The single database proves to be a bottleneck. The goal of this project is to replace the server-side module, partially or completely, in order to replace the single database with a peer-to-peer based system built on PIAX.

**PIAX**

PIAX is a peer-to-peer platform on top of which the DIPLOID project is being developed. It is written in Java, and offers its users a portable, flexible, scalable and powerful agent-based framework. Based on a SkipNet architecture, PIAX offers a distributed hash table, or DHT, and a novel technology called LLNet. Short for longitude-latitude network, it allows for agents to have one or more geographical locations on the world map. Area searches are supported of course, and it gives agents the power to find, share and reason about location-based information.
Although PIAX is powerful, it does contain some yet to be resolved issues. The main one is churn. Churn is the continuous process of nodes joining and leaving the network. When a node joins or leaves, the SkipNet has to be partially reorganized. During this process of reorganization the network may not be able to find all of the requested information. A solution is being worked on, so we will not try to solve this problem in this project.

PIAX is being developed at the CyberMedia Center at the University of Osaka, Japan, and is being distributed as open source. Future versions of PIAX, which are currently in development, it will be possible to assign a physical location to a peer or leave this undefined. A peer will have the capability to have the same physical location as the peer assigned to it, one or more logical locations or undefined. The flexibility of the network will be greatly increased by these options. With the release of the new version, within one or two weeks the architectural design might be updated to take full advantage of these new powerful features.

**SOAP**

SOAP, or Simple Object Access Protocol, is a system for remote method invocation using XML for serialization and usually HTTP as a carrier. Its use is widespread, and libraries exist for many programming platforms. Since we aim to support as many different of these platforms as possible, SOAP has become the obvious choice.

For this project we had two options of SOAP implementations. One was to use the Jetty web server which comes with PIAX, and write a servlet that can interface with the rest of the system. The other option was to use SoapRMI, a full featured SOAP implementation with a programming interface identical to that of java.rmi. Because of its simplicity and non-intrusiveness we went for this option.

A drawback of using SOAP is that it can cause quite some overhead. All the information has to be wrapped in XML, which then has to be sent over HTTP, which in turn has its own headers etc. It would therefore be a plus if we could keep the amount of SOAP operations at a minimum.
Design
Work division

Architecture

Peer - Agents
A peer is a computer or server that is connected to the PIAX-network and is uniquely defined by its peer id within the network. A peer runs one or more agents; each providing a service to the network. These agents have a unique reference id within a peer. Combined with its peer id, all agents are distinctive and can be relocated on the network.

Routing on the network is handled by the PIAX-framework; making the initial idea whereby applications connect to peers and/or agents using an URI-like schema no longer necessary nor possible. In the current version of PIAX, each peer has a location and all its agents share this location.

URI-like schema: piax://<authority>@<peer>:<agent>?<query>

In contrast with a traditional Hash table, addition or removal of one PIAX-agent responsible for storing a part of the LLNet causes nearly the entire key space to be remapped. In a distributed environment this results to bandwidth-intensive movement of the stored data from one node to another (Churn Resilience). Therefore this design makes a distinction between stable and unstable clients, whereby the latter don’t participate directly in the network but connect to it using a SOAP interface.

3 Implementation with the current version of PIAX would add greaten the complexity of the design.
Location based information

As the LLNet represents a plane, it is capable of storing geographically linked peers. All agents within a peer share its location. The information is stored, relocated, reallocated and removed using the LLNET capabilities of the PIAX-platform.

In figure $x_1$ the plane represents the world or a part of the world, whereas the dots are agents located in the LLNET skip graph. The function callMulti ($x$, $y$, $w$, $h$, kind, method, args) is used for executing range queries. The method call specified by method will be performed at all agents of kind kind on all peers in the area specified by ($x,y$) and ($w,h$). Possible examples of callMulti using Figure $x_1$ (see the blue square in the right-lower corner) are:

- $\text{callMulti}(X,Y,W,H, \text{green}, \text{weather}, [05/26/2006]) >>> \text{Rain}$
- $\text{callMulti}(X,Y,W,H, \text{green}, \text{rain}, []) >>> \text{TRUE}$
- $\text{callMulti}(X,Y,W,H, \text{green}, \text{avgWind}, [05/20/2006, 05/26/2006]) >>> 3,5$

The PIAX platform handles the call and returns the information to the client when finished. Clients connect to the network using a SOAP-interface.
A more schematic overview to the range query based architecture:

1. An unstable client sends a request to a stable entry in the network via a SOAP interface.
2. PIAX-platform looks up where data is stored
3. CallMulti is executed in requested range
4. Requested content is returned to client

User based search

When storing links to user based entries, the network used a Distributed Hash Table or DHT. In this table key to the data is the unique user id as it is used by TypeKey. As security is a major difficulty and
probably a project by itself, which this design does not try to address. All data is stored locally, and the all the network does is point to the data. Therefore a content provider should manage its own security and – by preference (can/should we enforce this, so authentication is approached in the same manner as the rest of the network?) – implement the TypeKey interface.

Examples of user based data:
- Buddy list
- Personal information (name, e-mail, telephone, …)
- Messages
- …

The algorithm is in essence the same as Location based search.

Authentication

![Diagram of authentication using TypeKey](image)

Since the two Hash tables only store links to the data, all actual data is kept outside the network. The network is only a mien or retrieving the information. Therefore authentication is not part of the network. Services requiring extra information can implement user validation using the TypeKey service. Also when locating user-information in the DHT, where entries are stored by distinctive TypeKey indexes, it will be necessary to first authenticate the user.

Authentication handling:

1. Glue code requests to authorize user
2. TypeKey responds with unique User Id
A possible front end application that uses the PIAX-based network, is the GMapWiki web application. When a user logs on to the service, she has to enter user name and password in order to retrieve personal information. The authentication is handled at the (web-) server side using Ruby, Python or another scripting language as glue code. The script will connect to the online TypeKey-service for validating the user. When the user is authenticated, a unique user id is returned by TypeKey, this id can be used to retrieve the stored user date from the Distributed Hash Table (DHT). An example of such data is the buddy list. For looking up geographically linked information, the LLNet is used. This can be done in combination with information retrieved from the DHT (e.g. : Buddy list for group entries) or the unique user id.
B. Plan of approach

Version: 1.6
Date:

Foreword

This plan of approach is part of the GMapWiki, a research project by Osaka University – CyberMedia Center. Goal is to provide the project leaders, team members and other readers insight into the content and planning of the given assignment.

The document should be used as a guide during the project. It describes the problems and the solution the project will solve, what should be done, what targets should be met and a brief planning for the several phases and targets of the project.

The authors would already like to thank some people for their much appreciated effort, assistance and interest to the project. First is Prof. Shimojo for assigning us. Dr. Teranis for his patience while helping and supporting in our work on his project; Ir. Sodoyer for reviewing our project goals and planning; and last but not least: our fellow students at the laboratory for making us feel right at home.

0. Overview

The GmapWiki-project usage is growing, threfore the current database implementation starts reaching it's performance limits. The assignment is to develop a scalable peer-to-peer solution to this problem. This document describes how we will get to this solution.

First we will study peer-to-peer systems in general and the current implementation for three weeks. After that we will start thinking about a better approach and how we will get to this solution for three weeks. Finally for 9 weeks will be worked on the designed system. The main goal of the project is to implement a scalable implementation. The design must be scalable enough to support load ballancing schemes.

1. Introduction

GMapWiki [http://gmapwiki.org] is an add-on to the GoogleMaps [http://maps.google.com] project developed by the CyberMedia Center at Osaka University, Japan. This web based application allows users to save geographically linked information and share their findings with the world (public), in a closed group or keep the stored data to themselves (private). As with all Wiki-applications; all information is entered, edited and deleted using the Wiki-standard [http://en.wikipedia.org/wiki/Wiki]. However, the reader should keep in mind that information can be kept private or in closed group.
Currently the database uses a single database to store all information. Although the system is still in an experimental face, this is proving to be a serious bottleneck in the system as the number of notes and users grows rapidly. For our bachelor project we received the assignment to speed up the current implementation of GMapWiki by distributing all data peer-2-peer wise over multiple systems. Here fore we will be using PIAX [http://www.ais.cmc.osaka-u.ac.jp/reserch_ubiq.html], a P2P agent platform developed at Osaka University.

Both Dr. Y. Teranishi and Ir. B.R. Sodoyer, respectively our Japanese Sensei and Dutch Mentor, are kindly requested to moderate this document as we – the authors – are looking forward to their highly valued feedback. Further we will report to Osaka University on weekly bases by a short presentation for our Sensei and fellow students and mail weekly progress reports to Delft University of Technology (TU Delft). This plan of approach will be adjusted weekly, based on the feedback of our Japanese and our Dutch mentors.

This document consists of the following sections: Project Assignment describes the changes to the current implementation of GMapWiki to obtain the desired result. The Approach-section contains a description of how to get the desired result. Project set-up and Terms will give an overview of how the organization of the project team is and under which conditions is worked. Planning will contain a description of the planning of the project. Finally, the last chapter contains recommendations for the future.

2. Project Assignment

As the GMapWiki-application is available to all Internet users; the growing popularity results in a sharp decline of performance. In the current implementation all data is stored on a central server; a heavy burden on scalability and thus identified as a bottleneck by Osaka University-researchers. Aim of this assignment is creating a P2P based data storage using Osaka University' PIAX-platform.

The main target of this bachelor project is to speed up performance. By an efficacious design the updated GMapWiki-application should provide scalability, data load balancing between the different P2P-servers (?) and outline a solid platform for future functionality. After adding the above mentioned functionality, it should be an issue to add more computer power to support the expanding group of users, geographically-linked notes and processing queries.

First there are a few technical problems which need to be solved. Currently there is prototype peer-to-peer implementation available, but as this implementation is far too slow for any practical use and can't communicate with the front-end application; the GMapWiki Homepage. Nevertheless we hope to research the application and learn for the strong points as from the weak. Thence standing/building on the shoulders of giants to design software and a (new) database system ready for the production environment, connected to the front-end code using PIAX-agents.

To summarize our priorities:

- The most important part of this project is to speed up the implementation of the storage system. Likewise it will be possible for more users to work with the application. We should think of ways to minimize the processing time when querying or adding notes to the system .
• Growth of the system should be supported by making it P2P.
• When a production ready and stable code-base is created; connecting the front-end o the newly created platform.
• Provide a platform for further implementation and new features.

Unfortunately, every project has its limits. Because this team has the strong desire to create a production ready web application, research to new technologies shall be limited. This will shift our focus to proven and well-known algorithms, which might cause a trade-off between scalability and load balancing.

3. Approach

The software design falls apart in the following three main problems: Setting up a distributed P2P database using PIAX, ensuring scalability and introducing load balancing. Although the research preceding to the writing of this document gave all team members an overall picture of the different approaches to P2P systems, most details of this outline are still open. The fine points will be studied prior to the prototyping and implementation of each sub-part in which this assignment falls apart. Progress shall be presented during the weekly presentations at Osaka University – CyberMedia Center and in briefly summarized to Delft University of Technology.

4. Project set-up and terms

4.1 Project set-up

The direct supervisor to GMapWiki is Dr. Teranis from Osaka University monitoring progress, provide the research group on planning and technical issues that might arise. For Delft University of Technology, the project is overseen by Ir. Sodoyer, making sure that the project meets all requirements demanded by Computer Science Bachelor project.

The project-team (In alphabetical order: Jeremy, Menno and Niels) should have or gain knowledge about P2P systems, data storage, object oriented software design, object oriented programming, planning. More specific: the project-team needs understanding of the PIAX-P2P platform, Knowledge of Java and basis understanding of the Ruby-scripting language to connect the implementation to the already existing front-end.

All members to the project-team will work 5 days per week in a 15 week-long schedule; that is without circumstances beyond control. In a case of force majeure the schedule should be modified conform a reasonable workload for the time the project-team sojourns in Japan.

During weekly meetings the team will report to Dr. Teranishi. Ir. Sodoyer will be sent weekly reports of this progress, this via e-mail. In this reports the project-team will consider the progress of the GMapWiki-project and an updated planning.

All project members have a workstation on which Ruby, Java and Java-script is supported. The workstations will have an Internet connection good for research and downloading documentation. Printers are if necessary at disposal.
In case of extra expenses of financial nature related to GMapWiki and the conducted research, the project team has the right to propose in advance an estimation for financial intermediation from the CyberMedia Center. The CyberMedia Center is entitled to refuse this or propose an alternative solution.

4.2 Terms

As required by Osaka University, the project members are obliged to be present in Osaka during the entire semester. They are expected to be at the CyberMedia Center student lab five days a week, from Monday till Friday, except at national holidays such as golden week. They will attend the weekly progress discussion meetings at the CyberMedia Center.

The project team will use a source code management system to commit the changes in the code. Changes in documentation will be distributed using email. To guarantee stability there will be issued nightly builds and all new written code will be automatically checked using JUnit test-cases.

While working at the lab, the project members will research, prototype, design, implement and test a new backend system for the GMapWiki project, based on PIAX P2P-technology. If the project goes smoothly and the project does not fall behind schedule, free time may be created to explore the country.

Osaka University will provide the project members with accommodations for the duration of the project. They will assist the project members in every matter such as getting around the city, opening a bank account, and other administrative problems.

Project team is allowed to attend the JB100 elementary Japanese course during lab hours.

Project members will not break the bicycles issued to them by Osaka University.

5. Planning

The project is separated in three parts: the analysis phase, the design/re-design phase, the development and test phase. Because our experience with peer-to-peer systems is quite limited, we need to do some research on several different peer-to-peer systems and techniques. The current implementation of the system is also unknown so we need to do some research on this system to before it can be changed. Because of these two quite unknown factors, we planned an analyses phase of three weeks.

After the analysis we want to redesign the old system and design the new (sub) system. In this phase we already know how the current system works and what the desired functionality of the new sub system should be. Because we don't have much experience with developing peer to peer based systems and because for the resulting system it would be desirable to have a stable prototype system, we planned in agreement with the CyberMedia Center a design phase spanning three weeks. After the design phase follows the development phase. Because of the stability-requirement we decided automatic test-cases should be developed. Therefore the development phase is also the test phase.
0. General Research
   o Research on P2P systems (week 1)
   o Writing plan of approach (week 2)
   o Delivering plan of approach (week 3)
   o Java-Practice (Programming exercise) (week 3)

1. P2P-database (providing scalability)
   o Research (week 4)
   o Architecture Design (week 5)
   o Class diagram Design (week 6)
   o Finishing RAD (week 7)
   o Implementation (week 7)
   o Documenting (week 9)
   o Delivering (week 10)

2. CRUD (Create, Read, Update, Delete) + Search operations
   o Design (week 11)
   o Implementation and testing (week 12)
   o Documenting (week 14)
   o Delivering (week 15)

3. Connecting code to front-end
   o Delivering (week 15)

4. Recommendations and reporting
   o Researching future recommendations (week 16)
   o Overall report on the bachelor project (Delft)
   o Presentation (Delft)

6. Quality Warrant

For Osaka University it is important that a scalable and distributed application is delivered. To make sure that all code and the overall implementation works as desired and to the expectation of a production ready system, the project-team considers it as highly important to design, write and run test classes and simulations. These simulations can be high- (different P2P clients communicating, exchanging data, etc) or low level (test methods at code or class level).

To assure high quality code, the development process should use latest techniques in software engineering. Previous written test cases should be able to run without error, after changing code. The practiced programming languages should be well known and studied on regular basis. When adding new features to the implementation, it is good practice to add test cases before developing the actual code. The author of a test case should strive to obtain the highest test coverage possible.

The code base should always be in an -almost- deliverable statue. To assure this we run a nightly build, in
which the test cases run as well. So when time runs short, a working system can still be delivered.

7. Other Plans

In the current project is mainly for a working, scalable, stable system, which is ready for a production environment. Which means we can't concentrate on many new cutting the edge features. In following versions there we think there can be worked on the following issues:

- More advanced Automatic outbalancing algorithms. When one peer has high load and another host doesn't, it should be possible to distribute the work over several peers automatically.

- Currently the front-end application is still running on a single system. This could become a bottleneck in the future. While it is possible to connect multiple web servers to the peer-to-peer system, the users should be automatically distributed over the multiple web servers.
C. Progress

Week 0  
08\(^{\text{th}}\) April 2006 – 14\(^{\text{th}}\) April 2006
- Departure at Brussels National Airport, Belgium
- Arrival at Narita (Tokyo) Airport, Japan
- Arrival at Osaka University Foreign Student House, inspecting and accepting room
- Introduction to Osaka University at IC Hall
- Arranging submission to Osaka University
- Introduction meeting with Prof. Shimojo en Prof. Teranishi
- Meeting with Prof. Teranishi, outlining general idea of the assignment
- Alien registration at Suita City Hall, opening obligatory bank account for paying rent, etc

Week 1:  
17\(^{\text{th}}\) April 2006 – 22\(^{\text{nd}}\) April 2006
- Studying P2P documentation and papers
- Studying GMapWiki
- Presentation:
  - Introducing ourselves
  - Introducing TU Delft

Week 2:  
24\(^{\text{th}}\) April 2006 – 28\(^{\text{th}}\) April 2006
- Presentation on PIAX by main developers
- Studying PIAX
- Studying P2P systems
- Presentation on “what we have learned” concerning P2P systems
- Writing plan of approach
- Research on Load Balancing
- “The Java practice”: Osaka University exercise, programming in JAVA

29\(^{\text{th}}\) April 2006 – 5\(^{\text{th}}\) May 2006
- Golden Week (Gōruden Wīku)
  Several Japanese holidays that occur during the first week of May.
  29\(^{\text{th}}\) April: Nature Day
  3\(^{\text{rd}}\) May: Constitution Memorial Day
  4\(^{\text{th}}\) May: Citizen’s Day / Nature Day

University closed for the week.
Week 3:
8th May 2006 – 12th May 2006
- Start writing requirements
- Meeting with prof. Teranishi on requirements
- Feedback on requirements
- Feedback plan of approach

Week 4:
15th May 2006 – 19th April 2006
- Start architectural design
- Meeting with prof. Teranishi on architectural design
- Researching JSON, SOAP, PIAX, etc
- Niels working from dormitory on PIAX-research due to flue

Week 5:
- Architectural design
- Presentation on Architectural Design
- Feedback on presentation
- Start class design
- Research framework designs
- Group meeting on overall progress

Week 6:
29th May 2006 – 2nd June 2006
- Finishing Architectural design
- Final feedback on architectural design
- Continuing work class diagram
- Start documenting class diagram
- Attending presentation of prof. Shimojo on role of CyberMedia Center

Week 7:
5th June 2006 – 9th June 2006
- Group feedback on class diagram
- Finishing documentation on class diagram
- Finishing RAD
- Overall evaluation
- Visiting INTEROP (http://www.interop.jp/imc/) with laboratory 7th till 9th June, Tokyo.
Week 8:
12th June 2006 – 16th June 2006

- Presenting RAD-document to Teranishi-san and UP2U weekly meeting
- RAD document approved by CyberMedia Center
- Dividing work on implementation
- Start implementing
- Continue to work on overall bachelor project documentation in spare hours

Week 9:
19th June 2006 – 23rd June 2006

- Continue implementation
  - Finished packages:
    - Data
    - Filter
  - Work in progress:
    - Peer
    - Sensor
    - Standalone
  - Problems
    - Testing SOAP using JUnit (priority: medium)
    - SVN web interface (priority: low)
- Weekly UP2U meeting
- Progress evaluation
- Continue to work on overall bachelor project documentation in spare hours

Week 10:
26th June 2006 – 30th June 2006

- Continue implementation
  - Packages:
    - Peer
    - Sensor
    - Standalone
- Weekly UP2U meeting
- Continue to work on overall bachelor project documentation in spare hours
- Friday: Welcome party for new students

- Waiting for PIAX-platform before implementing P2P façade class (placeholder code).

Week 11:
3rd July 2006 – 7 July 2006

- Start implementation web front-end (Ruby)
- Weekly UP2U meeting cancelled due to exams for the Japanese students
- Java-client front end (low priority)

- Waiting for PIAX-platform before implementing P2P façade class (placeholder code).

**Week 12:**
10th July 2006 – 14th July 2006
- Weekly UP2U meeting
- Continue implementing front-end (Ruby)
- Writing documentation on Sensor interface
- Seminar on speech fluid dynamics by CyberMedia Center

- Waiting for PIAX-platform before implementing P2P façade class (placeholder code).

**Week 13:**
17th July 2006 – 21st July 2006
- 17th July National Holiday
- Weekly UP2U meeting
- Connecting GMapWiki front end
- Testing newly arrived PIAX
- Lab trip to Kobe

**Week 14:**
24th July 2006 – 28th July 2006
- Adding PIAX to GMapWiki Storage platform
- Finishing Front end connection
- Continuo work on documentation
- Start working on presentation (draft version)
- Crash test sensor
- Weekly UP2U meeting

**Week 15:**
31 July 2006 – 4th August 2006:
- Weekly UP2U meeting
- Writing recommendations
- Finishing implementation
- Crash test
- Writing presentation
Week 16:
7th August 2006 – 11th August 2006

Planned:
- Monday: Final presentation (Osaka University – CyberMedia Center Version)
- Administration; returning keys to the dormitory and bicycles to IC-Hall
- Friday: Departure for Japan (Tokyo Narita Airport – Japan)
- Friday: Arrival in Europe (Brussels National Airport – Belgium)

Week 17:
September

Planned:
- Presentation Delft