MASTER'S THESIS 2010:XX

Augmented Reality on iPhone

Full implementation of a Client-Server Application.

JEAN-LOUIS GIORDANO

Department of Signals and Systems
Division of XXXXXXXXXXXXXXXXX
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2010

Augmented Reality on iPhone
Full implementation of a Client-Server Application.
JEAN-LOUIS GIORDANO

Master's Thesis 2010:XX

Signals and Systems
Division of XXXXXXXXXXXXXXXXX
Chalmers University of Technology
SE-41296 Göteborg
Sweden

Tel. +46-(0)31 772 1000

Cover:
Artwork of the application used on the AppStore, drawn by the author.

Reproservice / Department of Signals and Systems
Göteborg, Sweden 2010

Augmented Reality on iPhone
Full implementation of a Client-Server Application.
Master's Thesis in Communication Engineering
JEAN-LOUIS GIORDANO
Department of Signals and Systems
Division of XXXXXXXXXXXXXXXX
Chalmers University of Technology

Abstract

Bla bla.

Bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla.

Keywords: iPhone, Augmented Reality, Realtime, AppStore, Ruby, Erlang, Objective-C, Mnesia, Google Map

Contents

Acknowledgements

1. Introduction

Augmented Reality is a dream coming true. The concept itself is not brand new and has been developed and studied for many years, but modern technologies and devices are now allowing Augmented Reality to be implemented almost anywhere and be used on a daily basis, in a vast range of applications from advertising to medical assistance.

At the mean time, the last generation smartphones have been equipped with sensors that locates its users and provide new cleaver and innovative interactive experiences. The iPhone is one of those, and taking advantage of its hardware to build an Augmented Reality experience is an interesting challenge.

This master thesis focuses on the design of an application of Augmented Reality on iPhone 3G-S and its implementation from both a Client and Server point of view.

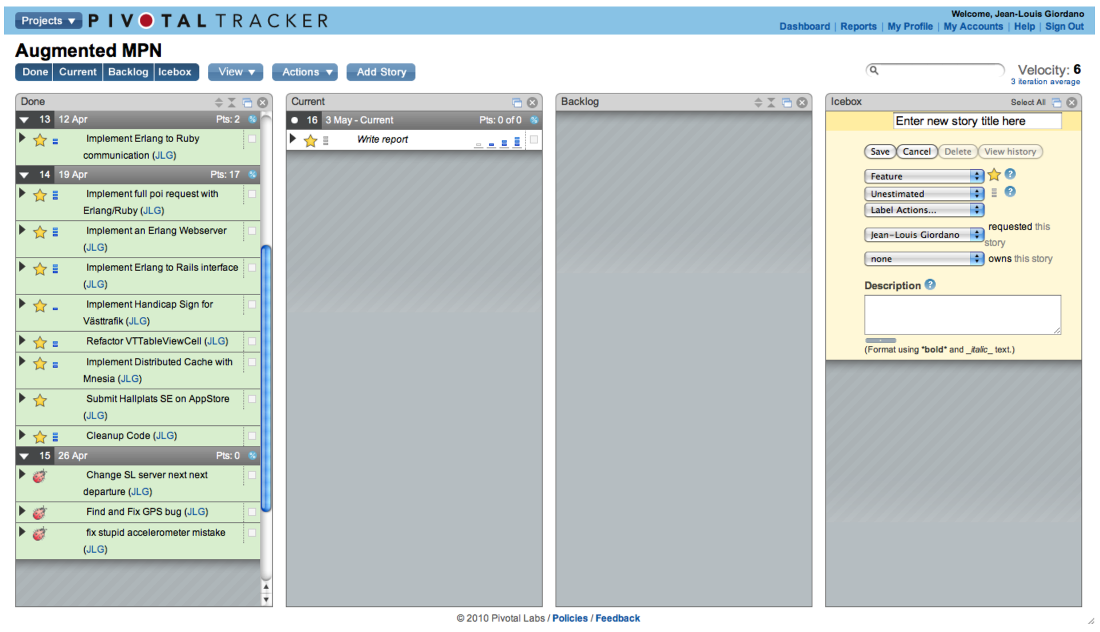
Work has been carried at ICE House AB, a development company for web enterprises.

At first an overview over the basic principles of Augmented Reality (Chapter 3) will help understanding what it is, how it works and how it can be used. Then we will have a look at the Client side of the application (Chapter 4) to see the definition of the application from a user's point of view, and we will go through the implementation on the iPhone to understand the underlying code structure. We will also have a look at the Server implementation (Chapter 5) to get an idea of the required architecture to provide data in real-time. Finally, we will describe the results achieved with the current implementation (Chapter 6), and a view on further improvements will finalize the thesis.

2. Methodology

2.1. Planning

Planning and task scheduling have been carried using Pivotal Tracker, an online agile project management tool. With this tool, one can estimate his working velocity to make planning decisions based on past performance. It also enable real-time collaboration so that everyone in the team is able to interact with the planning.



Augmented MPN

Done

13 12 Apr Pts: 2
Implement Erlang to Ruby communication (JLG)

14 19 Apr Pts: 17
Implement full poi request with Erlang/Ruby (JLG)
Implement an Erlang Webserver (JLG)
Implement Erlang to Rails interface (JLG)
Implement Handicap Sign for Västtrafik (JLG)
Refactor VTTableViewCell (JLG)
Implement Distributed Cache with Mnesia (JLG)
Submit Hallplats SE on AppStore (JLG)
Cleanup Code (JLG)

15 26 Apr Pts: 0
Change SL server next next departure (JLG)
Find and Fix GPS bug (JLG)
fix stupid accelerometer mistake (JLG)

Current

16 3 May - Current Pts: 0 of 0
Write report

Backlog

Icebox

Enter new story title here
Feature
Unestimated
Label Actions...
Jean-Louis Giordano requested this story
none owns this story
Description
(Format using *bold* and _italic_ text.)

© 2010 Pivotal Labs / Policies / Feedback

Figure 2.1.: Screenshot of the Pivotal Tracker tool

With this tool, a task is represented by a story. A story can be a new feature, a chore, a bugfix or a release date. The last one will appear as a deadline, whereas the others will be represented as a task to be carried out.

When creating a new story, one should evaluate the time to be spent on it. This helps the user to estimate his working velocity. A task should be defined such that it would not last longer than half a day.

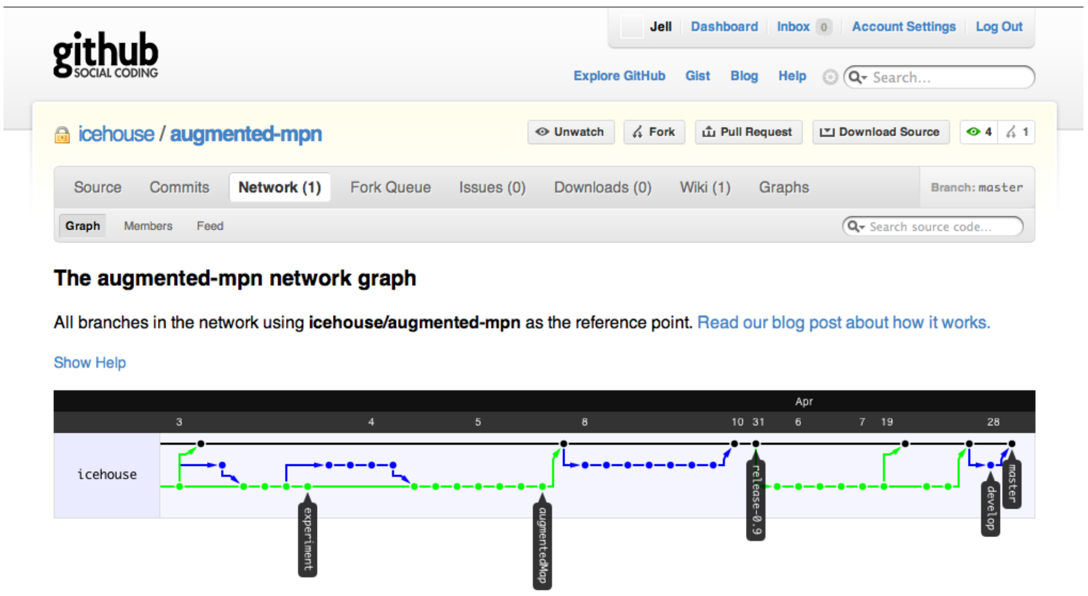
Once the story is added, it is put into an Icebox, which means it is saved but not added to the planning. This can be useful if the story describes an optional feature or idea. Once the user decide that the task should be carried out someday, the story is put in the backlog. Then when the user starts working on the story, it goes to the Current pipe, and when the task is achieved it is kept in the Done stage.

This is a really convenient tool, and it has been used from the very beginning of this project both as a reporting and scheduling system.

2.2. Version Control

In order to keep a safe and coherent development process, Git has been used as a version control system. Git is a fast and powerful tool that allows complex revision management, including branching and merging with distributed development.

The project is hosted on GitHub, an online Git repository that anyone in the team can access and fork.



The augmented-mpn network graph

All branches in the network using **icehouse/augmented-mpn** as the reference point. Read our blog post about how it works.

Figure 2.2.: Screenshot of the GitHub repository

The commits in the Git repository correspond roughly to the stories defined in the Pivotal Tracker (see previous section).

3. Basic Principles of Augmented Reality

3.1. What is Augmented Reality?

Augmented Reality is a term that is used to describe digital systems that display an artificial visual layer of three- or two-dimensional models upon our natural perception of reality in real-time. It aims at improving or completing our limited way of seeing the real world by presenting artificial elements that would otherwise be hidden to our senses.

The most commonly known Augmented Reality application is the television weather forecast system. In this system, a reporter is acting in a real environment in front of a blue/green screen, but the spectator watching his TV will see the reporter standing in front of an animated map: the content of the background consists of virtual data mapped on a real world object in real-time. This is Augmented Reality.

It is during the year 1992 that a Boeing researcher named Tom Caudell first defined "Augmented Reality" in a paper called "Augmented reality: an application of heads-up display technology to manual manufacturing processes" [Cau 92]. The term was used to describe a digital system that displays virtual graphics onto a physical reality.

Although the concept of Augmented Reality itself is older than its name, since the first Augmented Reality device was build and presented in a 1968 paper by Ivan Sutherland [Sut 68]. The system was designed to track the head position of the user to project before his eyes two-dimensional models in order to create an illusion of three dimensions. It relied on the idea that moving perspective images would appear in three dimensions even without stereo presentation, a principle which is called the "kinetic depth effect". In such a system, the user will perceive the object as three-dimensional only when he moves, otherwise the object will only be seen as planar.

It has to be noted that Augmented Reality differs from Virtual Reality. In Virtual Reality, there are no elements of Reality, as much as in Reality there are no elements of Virtuality by definition. In his paper called "Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum" [Mil 94], Milgram defines the Reality-Virtuality Continuum shown in Fig. 3.1.

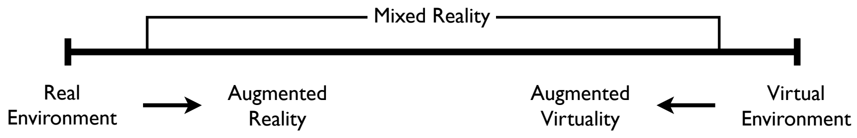


Figure 3.1.: Milgram's Reality-Virtuality Continuum [Mil 94]

As can be seen from this diagram, Augmented Reality is based upon reality and tends to Virtuality in an area called "Mixed Reality". It lies near the real world end of the line, which means the predominate perception is the real world. On the symmetric position, Milgram defines Augmented Virtuality. Behind this concept lie systems where real world items are added to a Virtual Environment, such as texture mapping on virtual objects in video games.

So the limits of Augmented Reality are defined by the furthest one can go at adding virtual data on a real world representation without depriving the user of its predominate perception of Reality.

3.2. How does it work?

In order to get an Augmented Reality application, one must integrate artificial objects into a real scene. Of course, such objects must be rendered to scale and at the correct position and orientation. To do so, the system requires to know the position of the camera or the user from the object. This is the core principle of Augmented Reality.

To solve this problem, there are two main approaches:

- Use sensors to know the position of the object from the camera

By the mean of tags on a real object, one can get a coordinate system on which to project a virtual model. Those tags can be visual or based on any technology that can be use to locate that object, for example a set of Infra-Red diodes.

- Use sensors to know the position of the camera in a known space

If the environment is known, i.e. the application is internally aware of the position of the object, or if the object is fully virtual and does not rely on any physical counterpart, one can evaluate the position of the camera in this environment and render the object accordingly. The position of the camera can be known by the mean of a large enough set of sensors that will be able to position it in the environment space.

Once the virtual environment is known, each virtual object must me rendered on top of an existing view by the mean of geometric projections that fit the estimated position of this object in the virtual space. The virtual object can also be projected onto a real world object such as a screen or an holographic system.

3.3. What is it for?

Applications are numerous and are invading a greater and greater number of domains: entertainment of course, with video games or virtual scavenger hunt, but it also has medical uses and presents various advantages for the industry.

In a recent web publication [Hay 09], an interesting study on Augmented Reality business model has been proposed by Gary Hayes showing its possible uses on the market. Figure 3.2 shows an attempt to categorize types of business dealing with Augmented Reality in order to identify opportunities in such field.

As can be seen from this graph, there is a very large amount of possible applications and it would be impossible to list them in an exhaustive manner. In this thesis, we tried to fit into the "Utility" business model, being the most promising one according to this graph (popular and high likely commercial value).

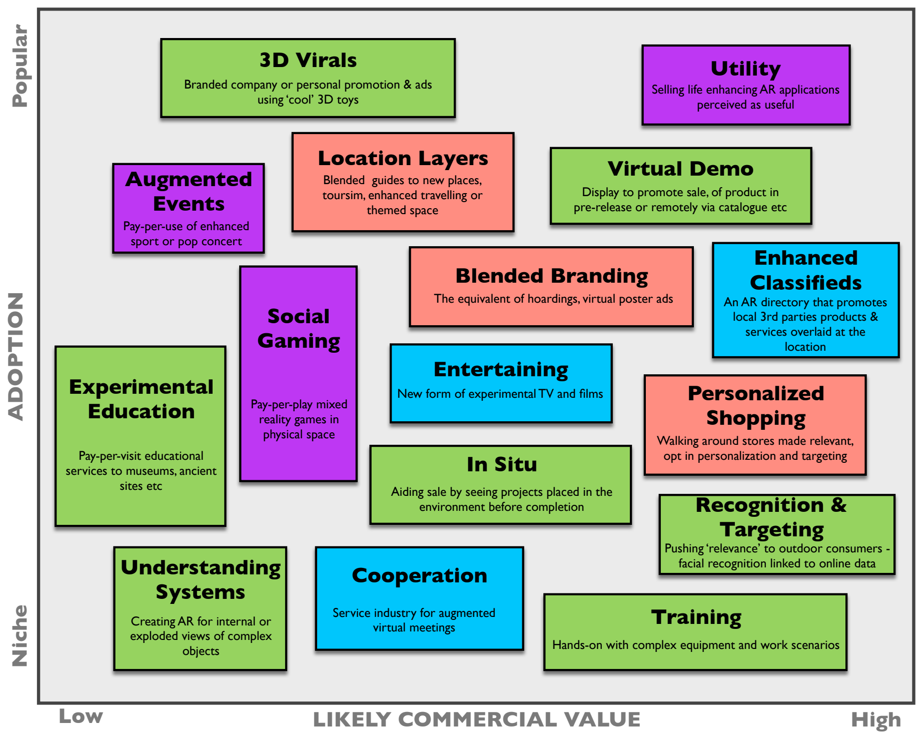

3D Virals
Branded company or personal promotion & ads using 'cool' 3D toys
Utility
Selling life enhancing AR applications perceived as useful
Augmented Events
Pay-per-use of enhanced sport or pop concert
Location Layers
Blended guides to new places, toursim, enhanced travelling or themed space
Virtual Demo
Display to promote sale, of product in pre-release or remotely via catalogue etc
Blended Branding
The equivalent of hoardings, virtual poster ads
Enhanced Classifieds
An AR directory that promotes local 3rd parties products & services overlaid at the location
Social Gaming
Pay-per-play mixed reality games in physical space
Experimental Education
Pay-per-visit educational services to museums, ancient sites etc
Entertaining
New form of experimental TV and films
Personalized Shopping
Walking around stores made relevant, opt in personalization and targeting
In Situ
Aiding sale by seeing projects placed in the environment before completion
Recognition & Targeting
Pushing 'relevance' to outdoor consumers - facial recognition linked to online data
Understanding Systems
Creating AR for internal or exploded views of complex objects
Cooperation
Service industry for augmented virtual meetings
Training
Hands-on with complex equipment and work scenarios
Popular
ADOPTION
Niche
Low
LIKELY COMMERCIAL VALUE
High

Figure 3.2.: 16 Augmented Reality Business Models by Gary Hayes [Hay 09]

4. On the Client Side

4.1. The iPhone

4.1.1. The Device

The iPhone is a last generation smartphone developed by Apple and was revealed for the first time in 2007. It features advanced functionalities such as multimedia support, internet access, video games and many more. Up to this day, forty-two millions iPhone have been sold across the globe.

Its main characteristic is that its User Interface (UI) is almost only based on two inputs:

- a large multi-touch screen
- a 3-axis accelerometers

As such, the iPhone does not possess any physical keyboard. Instead, a virtual keyboard can be summoned any time it is necessary, leaving a larger screen for applications.

The last generation of iPhone, the iPhone 3G-S, also includes a GPS, a compass, and a 3-megapixels camera, which seems to make it a perfect candidate for any Augmented Reality application. Unfortunately, the pragmatic access to the camera is not yet fully available, so image recognition is not yet an option, thought an upcoming version of its Operating System (OS) will allow the access to the video data of the camera.

In order to install new applications on an iPhone, one must download it from the AppStore, the internet-based retail store from Apple for iPhones, iPods and the iPad. Apple having a highly proprietary policy on its product, this is actually the only way to distribute iPhone applications.

4.1.2. The Development Tools

To develop an application for iPhone, a specific Framework is required. We will have a look at its nature in this section.

Objective-C

First of all, any software developed for iPhone must be programmed in Objective-C, although it is possible to call C and C++ functions from code.

Objective-C is an Object-Oriented (OO) reflexive programming language build upon the C language. It is comparable to C++ from this perspective, but differs greatly in many ways, especially in its dynamic message passing system, in being weakly typed and in being able to perform dynamic loading. In its latest version, Objective-C also features a Garbage Collector which subtracts the programmer from advanced memory management considerations. Unfortunately, this feature is not available for iPhone development.

As mentioned above, one of the interesting features of Objective-C is its message passing in lieu of method call for an object. In Objective-C terminology, calling a Method from an Object consists instead of sending a Message to a Receiver. The following example send a message ”say” with parameter @”hello world!” to the receiver named ”receiver”, and the answer ”result” is of type ”BOOL”.

```
BOOL result = [receiver say:@"hello world!"];
```

This is very close to the elegant Smalltalk messaging system. In practice, the Receiver is an object ”id”, which is similar to an anonymous pointer to an object. When messaging a receiver, the object sending a message does not need to know if the object it is sending to will be able to handle its request or not. This is really convenient since the sender does not need to be aware of the receiver. This system allows complex interactions between object.

iPhone SDK

In order to compile code for the iPhone, the use of Xcode as Integrated Development Environment is almost unavoidable. Apple has a highly proprietary approach for its products, and programming for an Apple environment is much restrictive to this regard. Fortunately, Xcode and the set of tools provided by Apple offer a great comfort of use in many cases, especially for debugging, or for creating interfaces with the use of the Interface Builder (IB).

Once Xcode is set up, the iPhone Software Development Kit must be installed on Xcode. The iPhone SDK takes advantage of all the features of Xcode, including its set of external tool to the largest extent. After being registered to the iPhone developer’s program, Xcode can also be linked directly to an actual iPhone device in order to test an application in real-time with any monitoring tool available.

This SDK contains Application Programming Interfaces (API) to all the accessible libraries of the iPhone, including Cocoa. Unfortunately, many functionalities such as the access to raw data from the video camera are not part of a public API, so their libraries are considered as not accessible and therefore their use are forbidden by Apple.

4.2. The Application

4.2.1. What it does

This application has been designed for travellers in Sweden that are looking for a nearby public transport station. Augmented Reality makes it very intuitive to find a nearby stop: the user just has to "look around" with his iPhone, and the closest stops will appear at their positions. When the user holds his iPhone in one direction, a virtual Map is projected on the estimated ground, and on it a set of three-dimensional pins indicating nearby stops. All this is done in realtime and fits the position of the iPhone with six degrees of movements.

To make it even easier for the user, a flat Google Map is available when the device is held horizontally. The flat map also displays pins representing the nearby stops. The Map is always oriented in the direction of the user for more convenience. This can be useful when directions are required.

If the user press a pin in either Augmented Reality or Flat Map, a bubble will appear with the stop name, its distance from the user and the Bus/Tram/Boat/Subway line numbers it serves. If this stop is interesting to him, the user can check the next departures by pressing the arrow in the bubble.

The application flips horizontally and displays the next departures in a scrollable view, with indexes on the side for quick navigation. This is a really efficient way to check for the next departures: only two "clicks" are required to access the desired data.

Augmented Reality makes the application very interesting for people that are uncomfortable at reading maps when discovering a new city.

But this application has also been designed for people that are already familiar with the city they visit. Simply by pointing at a stop they are interested in, they can get the next departures without wasting their time by going to a stop in order to consult its timetable.

For now, the application is available for Göteborg and Stockholm, with their respective public transport companies Västtrafik and Storstockholms Lokaltrafik. But additional providers could be added later on.

The application is available in English, Swedish, French and Chinese and can be downloaded from the AppStore under the name ”Hållplats SE”.

4.2.2. How it works

The application takes advantage of the GPS capabilities of the iPhone to locate the user on a map. Then thanks to the compass, we are able to estimate the direction he is looking at, and the 3-axis accelerometer allows us to evaluate the angle at which he holds his phone. Note that this application requires a GPS and a compass, and therefore is only compatible with the iPhone 3G-S.

Built-in APIs makes it easy to access the accelerometer, compass and GPS data. When an update to either of these sensors is generated, a message is sent to their respective delegates in a way that could be compared to Events in Java.

Once the position of the user is known, a request is made over the internet to determine the bus stops that are close to him. The answer to this request will be a list of stops with their names and locations together with a forecast list for each of them.

An item in the forecast list is composed of a line number, a destination and a set of attributes indicating the time of departure plus some information on the quality of the travel.

Once the positions of the bus stops are known, we can project them in the virtual space according to the user's coordinates, heading direction and phone holding angle. Our implementation of Augmented Reality hence follows the second method defined in Section 3.2: we use sensors to know the position of the camera in a known space.

To cope with the precision errors of acceleration and heading measurement, a buffer is used to get the average values on an arbitrary period of time before updating the knowledge on the user's position. Animations are also used to make transitions between two views appear smoother.

4.3. Implementation of the Augmented Reality View

4.3.1. View Hierarchy

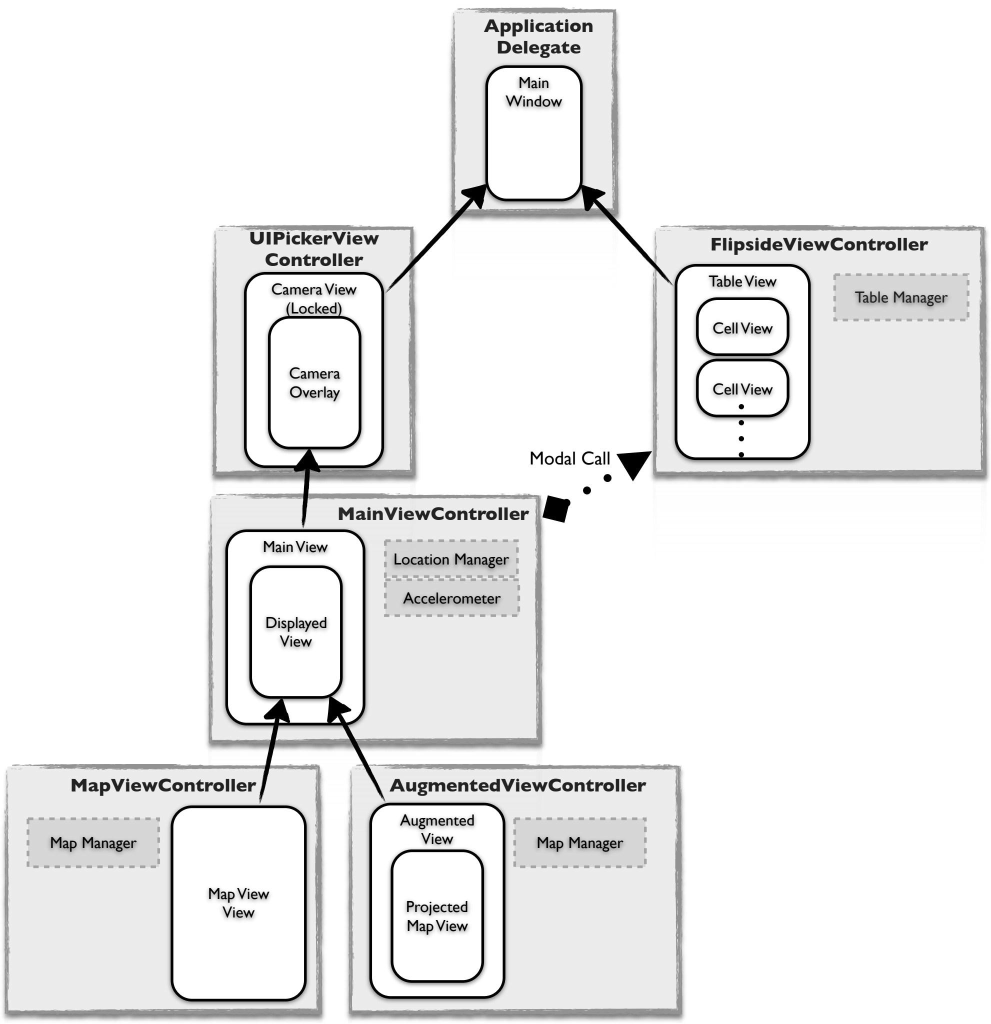


Figure 4.1.: Scheme of the View Hierarchy of the application

When developing for iPhone, the Model-View-Controller (MVC) is the best architecture to go with. There are well defined classes for views and controllers in the Cocoa environment, so we took advantage of that when designing our application.

When using the Cocoa environment, each view must be a subclass of the UIView class, and each controller of the UIViewController class. A view is automatically linked to a view controller.

A view can contain other views that will be considered as its subviews, and each view has a frame in which it renders itself. Views can have transparent or semi-transparent backgrounds, so it is possible to create a complex view hierarchy without hassle.

Figure 4.1 shows the view hierarchy for our application.

In order to render a view on screen, one must define an application window that is a subclass of UIWindow and that will take care of rendering every visible view on the iPhone's screen. By definition, a UIWindow must be the higher node in any view hierarchy.

Before further considerations, one should know that there is a serious restriction on the iPhone SDK up to the version 3.2 which prohibits to insert a camera view (UIImagePickerView) as a subview to a UIView. Instead, it must be directly put into a UIWindow. Since we want to use the camera in our application, we have no choice but to put it on top of our view stack. This is semantically very wrong, but there are no alternatives left here by Apple.

Furthermore, the camera view itself is locked, so we can't help but use a view called ”Camera Overlay” to display any view on top of it. That is ugly and clearly opposed to the concept of our application, but again there is no other way to do it.

As a result, the camera is always rendering in the background of our application, and everything else is rendered on top of it.

Now we have a window and a background with a camera view, and we want to be able to switch between a Map View and an Augmented Reality View. To do so, we implement a Main View that will display either of them according to the iPhone position. In order to avoid redundancy, all the data of the application is handled by the Main View Controller and dispatched in its subview, be it a Map View or an Augmented View. The same goes for the acceleration and location data.

To present the forecast of a stop, a ”modal view” is opened by the Main View Controller. The display of a modal view will replace the content of the actual window by a temporary view. When dismissing this view, the previous content of the window is loaded again and the application continues.

This modal view is a table view itself composed of view cells that are respectively subclasses of UITableView and UITableViewCell. Those view cells are used to display the forecasts list for a bus stop.

4.3.2. Positioning within 6 degrees of freedom

In our application, we need to locate the camera of the iPhone within 6 degrees of freedom:

- Latitude (φ)
- Longitude (λ)
- Altitude
- Azimuth (θ)
- Roll angle (α)
- Pitch angle (β)

Those degrees of freedom are illustrated in Figure 4.2.

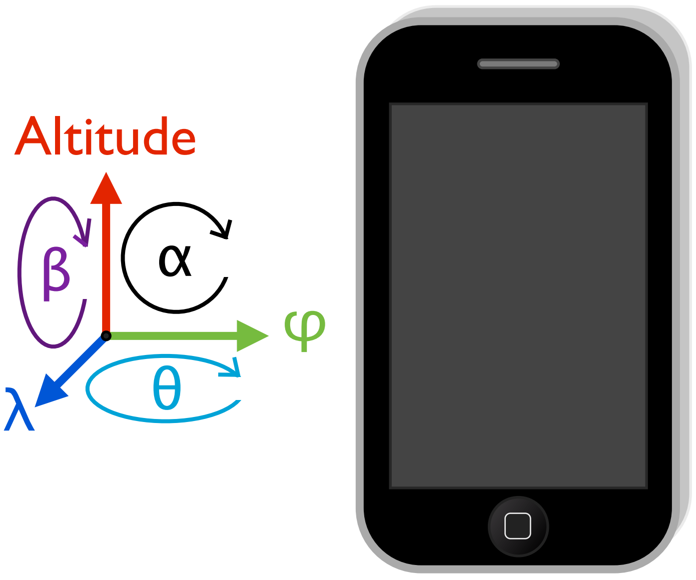


Figure 4.2.: the 6 degrees of freedom of the iPhone

The GPS directly gives us the Altitude, Latitude and Longitude of the user whereas the compass gives us his Azimuth. This directly takes care of 4 out of the 6 degrees.

But thanks to the 3-axis accelerometer, the roll angle and pitch angle can easily be computed by the mean of simple trigonometry as shown in Figure 4.3

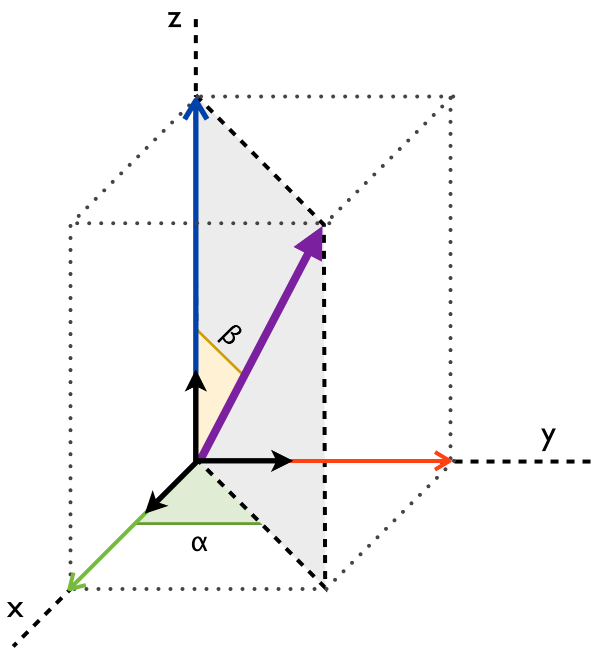


Figure 4.3.: Projection giving the Roll and Pitch angles, respectively α and β

To compute α and β, we use the following equations:

$$\alpha = \text{atan2}(x, y)$$

$$\beta = \text{atan2}(\sqrt{y^2 + x^2}, z)$$

Where x, y and z are respectively the accelerations values on the X-, Y- and Z-axis an atan2(a,b) is a function that gives the angle in radians between a vector (a , b) and the X-axis.

4.3.3. 3D Projection

Now that we know the position of the camera relatively to the object we want to display, we must use our knowledge to render the Augmented View.

Rendering the ground Map

First, we project a Map on the plane defining the ground. The iPhone has an API to perform 3D projections, by the mean of layer transformations. To apply a transform to a layer, one must provide the transform matrix corresponding to the desired projection.

In this case, the transformation will be a rotation of $\psi = \pi/2$ on the X-axis and another θ corresponding to the Azimuth on the Y-Axis, followed by translations t_y and t_z on the Y and Z axis to give depth to the view, which gives the following matrix:

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta & -\sin\theta/e_z \\ \sin\psi\sin\theta & \cos\psi & \sin\psi\cos\theta & \sin\psi\cos\theta/e_z \\ \cos\psi\sin\theta & -\sin\psi & \cos\psi\cos\theta & \cos\psi\cos\theta/e_z \\ 0 & t_y & t_z & t_z/e_z \end{pmatrix}$$

where e_z is the distance of the user from the projection plan. Since we have $\psi = \pi/2$, the transformation matrix becomes:

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta & -\sin\theta/e_z \\ \sin\theta & 0 & \cos\theta & \cos\theta/e_z \\ 0 & -1 & 0 & 0 \\ 0 & t_y & t_z & t_z/e_z \end{pmatrix}$$

Rendering the bus stops

Once the map is projected on the proper plane, we can project the buttons representing the bus stops. First, we need to compute the distance and the azimuth of the stop relatively to the camera.

To compute the distance d, we use a simplified version of the great-circle formula as shown in Equation 4.1 where φ_s and λ_s are the Latitude and Longitude of the standpoint (camera position), φ_f and λ_f the ones of the forepoint (bus stop location) and R is the average radius of Earth ($\approx$ 6371km).

$$d = R \times \arccos\left(\sin\varphi_s \sin\varphi_f + \cos\varphi_s \cos\varphi_f \cos(\lambda_f - \lambda_s)\right) \tag{4.1}$$

To compute the relative azimuth θ_r, we use the formula shown in Equation 4.2. In this formula, atan2(x,y) is a function that gives the angle in radians between the vector (x , y) and the X-axis.

$$\theta_r = \text{atan2}\left(\sin(\lambda_f - \lambda_s)\cos\varphi_f\,,\cos\varphi_s\sin\varphi_f - \sin\varphi_s\cos\varphi_f\cos(\lambda_f - \lambda_s)\right) \tag{4.2}$$

Now that we have the distance and the azimuth of the stop relatively to the camera position, we can use them as polar coordinates with the centre of the system being the projection of the camera point on the map plane. To project it on the view, we simply have to convert those coordinates to the Cartesian coordinate system of our view, which gives the following vector:

$$\begin{pmatrix} d\cos\theta_r \\ t_y \\ t_z + d\sin\theta_r \end{pmatrix}$$

The transform matrix is in that case really easy since it only consist of 3 translations, which gives the following projection matrix T:

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/e_z \\ d\cos\theta_r & t_y & t_z + d\sin\theta_r & (t_z + d\sin\theta_r)/e_z \end{pmatrix}$$

But there is a slight problem with this transform: one might end up with a projection that entirely fills the screen if the stop is too close from the camera. In order to avoid this situation, we must scale down the projection if it would appear too large. We can check that by looking at $T_{4,4}$ which gives in our case the inverse scale applied to the object during its projection.

If $T_{4,4} < 1$, then we must apply a scale to the transform matrix. Let's define the scale factor χ such as:

$$\chi = \begin{cases} 1 & \text{if } (t_z + d\ \sin\theta_r)/e_z \leq 1 \\ (t_z + d\ \sin\theta_r)/e_z & \text{otherwise} \end{cases}$$

Therefore, the new transform matrix T' is:

$$T' = \begin{pmatrix} \chi & 0 & 0 & 0 \\ 0 & \chi & 0 & 0 \\ 0 & 0 & 1 & 1/e_z \\ d\ \cos\theta_r & t_y & t_z + d\ \sin\theta_r & (t_z + d\ \sin\theta_r)/e_z \end{pmatrix}$$

Taking into account the Roll and Pitch angles

So we are able to render the map and the stops on it according to the latitude ϕ, the longitude λ and the azimuth θ of the camera, we now need to take into account the Roll angle α and the Pitch angle β.

To make everything simpler, the map and stops views are rendered in a container view which is then transformed according to α and β. This also makes the application more orthogonal since α and β are know via the accelerometer whereas the other parameters are known via the GPS.

What we want is to transform the container such as it is rotated according to α on the XY-plane and translated according to β on the Y-Axis. Rotation on the XY-plane is straightforward, and the translation is given by this formula:

$$t_{container} = \frac{\text{SCREEN WIDTH}}{\sin(\text{CAMERA SPAN ANGLE})} \cos\beta$$

This gives the following transform matrix:

$$\begin{pmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & t_{container} & 0 & 0 \end{pmatrix}$$

Summary

To sum up, we project the map in the container view using this projection matrix:

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta & -\sin\theta/e_z \\ \sin\theta & 0 & \cos\theta & \cos\theta/e_z \\ 0 & -1 & 0 & 0 \\ 0 & t_y & t_z & t_z/e_z \end{pmatrix}$$

Then we project each stop in the container view using this matrix:

$$\begin{pmatrix} \chi & 0 & 0 & 0 \\ 0 & \chi & 0 & 0 \\ 0 & 0 & 1 & 1/e_z \\ d\ \cos\theta_r & t_y & t_z + d\ \sin\theta_r & (t_z + d\ \sin\theta_r)/e_z \end{pmatrix}$$

And finally we project our container view in its parent view using this matrix:

$$\begin{pmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & t_{container} & 0 & 0 \end{pmatrix}$$

5. On the Server Side

5.1. Architecture

In order to retrieve information about nearby stops, the client must request a remote source of information by calling a server. The server that has been implemented in our case acts as a middleware between the public transport providers (Västtrafik and SL) and the client. By doing so, requests to the different providers are transparent to the client.

The server processes a request for nearby bus stops as follows:

1. receive a request with the user's coordinates (Latitude and Longitude),
2. find the 10 nearest bus locations from the given coordinates in its SQLite database,
3. for each of these stops, launch a light-weighted process that:
   a) if valid cached data is available, returns this data as forecast
   b) otherwise, fetch forecast data from the stop's provider
4. once each forecast is obtained, parse the result into JSON format,
5. finally, send a reply to the client containing the JSON data.

The application is hosted on a Yaws web server installed on a Debian (Ubuntu) virtual machine from the Amazon cloud web services. Yaws is the Erlang alternative to the Apache web server, and as such offers high availability and high scalability.

In order to handle high quantities of lightweight processes, the best solution was to use Erlang. Erlang is a functional programming language especially designed to handle high concurrency and distributed systems. A large part of the application is therefore implemented in Erlang in order to take advantage of this.

Figure 5.1 shows a simplified request processing on an UML sequential diagram without the caching system, the grey area being our server.

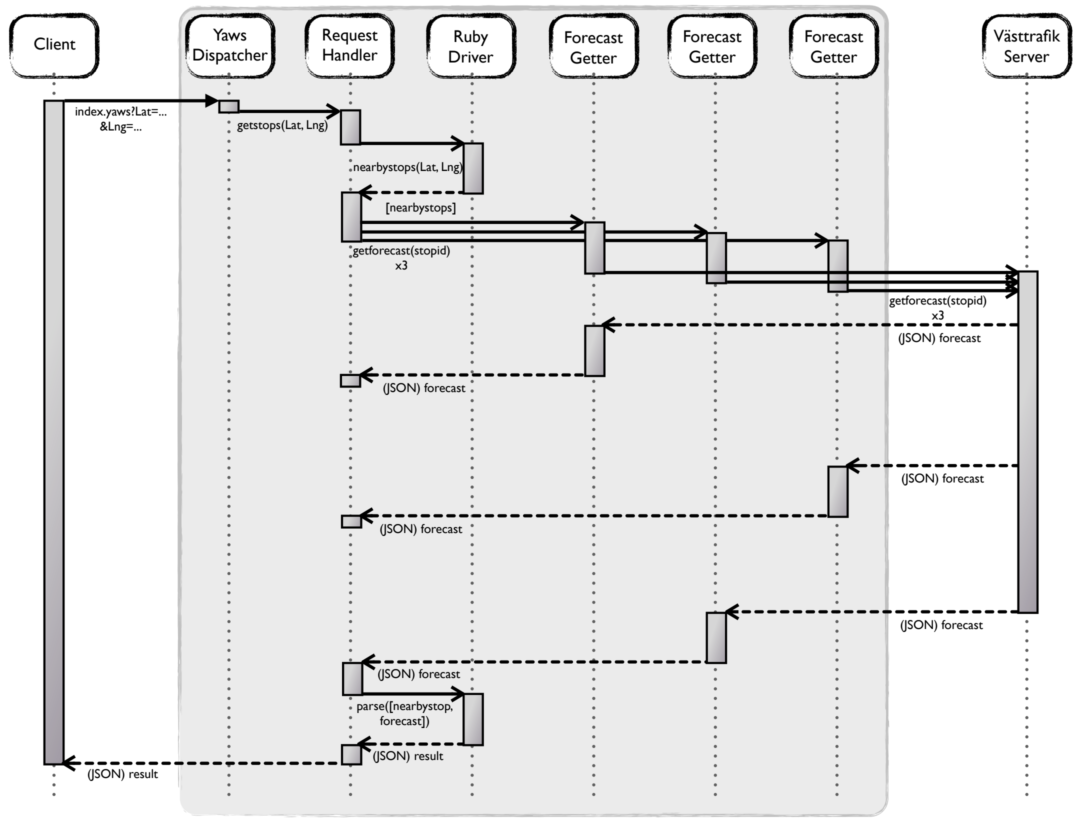


Figure 5.1.: UML Sequential Diagram of a simplified request

This diagram follows the classic UML notation, so plain arrows represent synchronous calls, stick arrows represent asynchronous calls, and dashed arrows represent return values.

But to keep the application easy to maintain and update, the more complex tasks are implemented in Ruby. Ruby is a modern dynamic scripted language, and has been designed to make the developers happy. As such, it is a powerful tool to implement complex algorithms painlessly. The ruby part is hosted on a rails web server, and communicates with the Yaws server through Erlang Ports via the Erlang Binary Protocol.

Figure 5.2 shows the details of the server's architecture.

In this figure, each circle represents an independent process and the arrows indicate message passing between processes. The dashed rounded rectangles represent databases servers, SQLite for Ruby and Mnesia for Erlang. Black triangle represent communication ports.

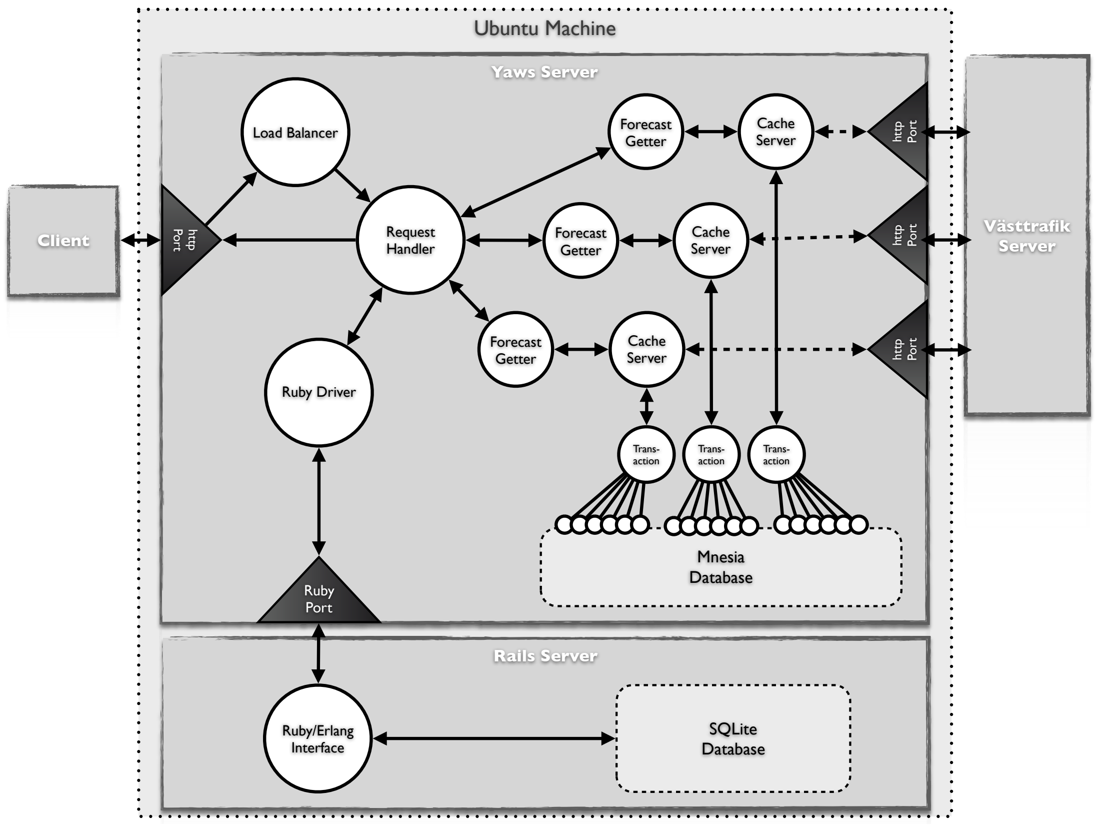


Figure 5.2.: Scheme of the Server Architecture

5.2. Implementation

5.2.1. Erlang

Erlang is used to dispatch the lightweight processes, to perform the distant requests to Västtrafik and Storstockholms Locaktrafik, and also to take care of data caching via a Mnesia database.

Concurrent Programming

As mentioned in the previous section, Erlang is a functional language, and as such differs greatly from classic programming paradigms. In Erlang, functions are treated in a pure mathematical way: variables are immutable and functions have no side effects. This may seems like a very handicapping restriction, but that is what makes Erlang so powerful at easily handling thousands of concurrent processes without having problems of dead locks, starvation nor memory corruption.

Being a functional language, Erlang allows function recursion without much overhead thanks to tail recursion optimizations. It also feature strong pattern matching capabilities. The piece of code shown in Figure 5.3 is an implementation of the factorial function programmed in Erlang.

```
-module(example).
-export([factorial/1]).

factorial(0) ->
    1;
factorial(N) ->
    N * factorial(N-1).
```

Figure 5.3.: Implementation of the factorial function in Erlang

From this snippet we can see that the Erlang implementation is very close to the mathematical definition of factorial:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

This is made possible thanks to pattern matching, and it is efficient thanks to tail recursion optimizations.

But the real power of Erlang reside in its messaging system that allows complex asynchronous calls. When a process sends a message to another process, the message is put into the receiver's message box to be treated whenever it is available. By doing so, message passing is a totally non blocking procedure.

Figure 5.4 shows a simple message passing procedure between two processes. The "spawn" command spawns a new process with the function passed as its argument, then returns its Process ID (PID). To send a message to a process, one must use the "!" command preceded by the PID of the receiver and followed by the message to be sent. The "self()" function returns the PID of the current process, so if an answer is required from the contacted process, one can pass its own PID along with the message it sends. The "receive" bock is defined to handle received messages matching the supported messages.

Erlang has many more features that will not be treated in this paper such as list comprehension or hot code swapping, but more information can be easily found on the Internet [Erl 10].

```
-module(example).
-export([start/0, ping/0]).

ping() ->
  receive
    {ping, Pong_PID} ->
      io:format("Ping!~n", []),
      Pong_PID!pong,
      ping()
  end.

start() ->
  Ping_PID = spawn(example, ping, []),
  Ping_PID!{ping, self()},
  receive
    {pong} ->
      io:format("Pong!~n", [])
  end.
```

Figure 5.4.: Simple example of message passing in Erlang

Port Communication

In our implementation, we need Erlang to communicate with Ruby. To do so, it is possible to open a ”Port” in Erlang that will maintain a link to an external driver, in our case a script on a rails server (See Figure 5.5). The Port will be kept open as long as a process is linked to it, so in our case we spawn a process that will act as a server to the Ruby port.

```
start_driver() ->
  Cmd = "rails runner ./lib/echo.rb",
  Port = open_port({spawn, Cmd}, [{packet, 4}, nouse_stdio,
            exit_status, binary]).
```

Figure 5.5.: Example of Port opening with Erlang

A Port is transparent in Erlang, so communicating through it is equivalent to talking to a local process. This is actually very powerful, because Ports are typically bridges between servers, so distant and local process communications are perfectly identical.

Mnesia

The standard database to be used with Erlang is Mnesia. With Mnesia, Erlang records represent key-value tuples where the value can be any data structure, from simple integer to Erlang lambda functions.

The query language to the database is Erlang itself and not a third party language such as SQL, which enable all the features of Erlang within the request procedure: pattern matching and list comprehension among others. To perform a query, one must perform a ”transaction”. A transaction will ensure that any read/write procedure to the database is performed properly. Transactions can easily be nested and can also be distributed on different process/servers transparently. A simple query example is shown in Figure 5.6.

```
all_females() ->
  F = fun() ->
      Female = #employee{sex = female, name = '$1', _ = '_'},
      mnesia:select(employee, [{Female, [], ['$1']}])
      end,
  mnesia:transaction(F).
```

Figure 5.6.: Simple example of query to Mnesia Database returning the names of all female employees

Our implementation of the cache server uses Mnesia to store cached data. The database only has one table containing the pairs of key-values corresponding to ”url”, ”data” and ”timestamp”. The URL is used as an index for queries to the database, and the timestamp is here to check when the cached data should expire. The full implementation is shown in Appendix B.

More information about Mnesia can be found on the Internet [Mne 10].

5.2.2. Ruby

Required Setup

Ruby is a modern dynamic object-oriented programming language in which everything is an object, exactly as the programming language Smalltalk from which it is partly inspired. It has been designed to follow the least surprise principle during development: code should never behave in an unexpected way. It is a powerful language for writing scripts and developing applications in a very productive manner.

Ruby comes with a package manager called RubyGems that provide a standard distribution system for libraries. A Ruby library is called a Gem.

Among those Gems we find Rails, a framework that allows to deploy web applications using Ruby. When creating a Rails application, a web server is packed in, and that is what we have used in our project. We worked with Rails 3.0.0beta3, the edge version of the framework, to deploy our Ruby application.

```
require 'rubygems'
require 'erlectricity'

receive do |f|
  f.when([:echo, String]) do |text|
    f.send!([:result, "You said: #{text}"])
    f.receive_loop
  end
end
```

(a) Ruby Side

```
-module(echo).
-export([test/0]).

test() ->
  Cmd = "ruby echo.rb",
  Port = open_port({spawn, Cmd}, [{packet, 4}, nouse_stdio,
                                  exit_status, binary]),
  Payload = term_to_binary({echo, <<"hello world!">>}),
  port_command(Port, Payload),
  receive
    {Port, {data, Data}} ->
      {result, Text} = binary_to_term(Data),
      io:format("~p~n", [Text])
  end.
```

(b) Erlang Side

Figure 5.7.: Simple example of communication between Erlang and Ruby using Erlectricity

An other Gem was require to implement an interface with Erlang, so we used Erlectricity [Fle 09]. This library enable Ruby to send messages via the Erlang Binary

Protocol, making communication between the two languages transparent. Figure 5.7 shows a simple implementation of communication.

One problem that we faced when communicating between Ruby and Erlang was when dealing with strings: Erlang can only handle Latin-1 encoded strings, whereas Ruby manipulates UTF-8 strings by default. So whenever a string is sent to be treated in Erlang, it has to be encoded in Latin-1 first, and when a string is received by Ruby, it has to be converted to UTF-8.

It has to be mentioned that Erlang and Yaws can handle strings in an arbitrary encoding under binary form, so when answering to the client the reply is actually UTF-8 encoded.

Database Management

Now that we know the required setup for our Ruby Implementation, let us have a look at what we want Ruby to do for us.

In our server application, Ruby is used to implement the SQL queries to the database and to maintain it. The SQL database contains information on all the bus stops for each provider. Each entry in the database has the following attributes:

•	stop_name	:	the name of the stop
•	lat	:	the Latitude of the stop
•	lng	:	the Longitude of the stop
•	provider_name	:	the name of the stop's provider (Västtrafik or SL)
•	stop_id	:	the id of the stop as defined by its provider
•	timestamp	:	the timestamp of the stop creation
•	id	:	a unique ID for indexing

This database is populated by a background process that fetch data from the providers on monthly basis (it is not that often that new stops are build or that old one are destroyed). Updates are performed via requests to the webservices of the providers.

The "lat" and "lng" attributes are the key entries allowing geographic search in the database, but finding nearby points from a database can be a tricky problem. In our case, we perform queries iteratively as shown in Figure 5.8 until a satisfying number of stops are returned.

The first step consist of a SQL query is of the following form:

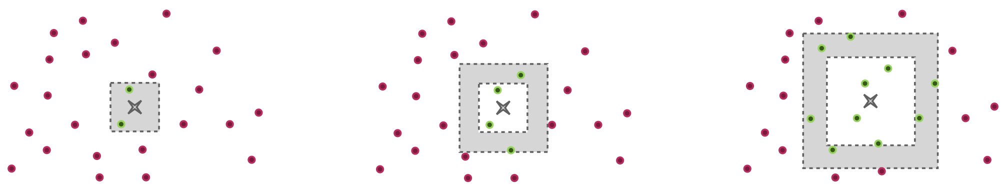

(a) Step 1 (b) Step 2 (c) Step 3

Figure 5.8.: Scheme of the SQL queries on the Ruby side

```
SELECT * FROM bus_stops WHERE lat >= lat1 AND
                              lat <= lat2 AND
                              lng >= lng1 AND
                              lng <= lng2
```

This will return all the bus stops within the area defined by the square defined by:

$$[lat_1, lat_2] \times [lng_1, lng_2]$$

The retrieved stops are stored in a bus list. The following steps produce queries defined as follow:

```
SELECT * FROM bus_stops WHERE lat >= lat1 AND
                              lat <= lat2 AND
                              lng >= lng1 AND
                              lng <= lng2 AND NOT
                                    (lat > lat3 AND
                                     lat < lat4 AND
                                     lng > lng3 AND
                                     lng < lng4)
```

This will return all the bus stops within the area defined by the area defined by:

$$[lat_1, lat_2] \times [lng_1, lng_2] \cap \overline{(lat_3, lat_4) \times (lng_3, lng_4)}$$

The retrieved stops are added to the stop list until the list is large enough to proceed. In our case, we keep stop the algorithm when the stop list has more than 10 entries.

Once we get a list of stops within an arbitrary large area, we need to sort those stops by distance in order to keep only the 10 closer ones. The distance d between the user's location and a stop is computed according to a simplified version of the great-circle distance formula given in Equation 5.1, where φ_s and λ_s are the Latitude and Longitude of the standpoint, φ_f and λ_f the ones of the forepoint, and R being the average radius of Earth ($\approx$ 6371km).

$$d = R \times \arccos\left(\sin\varphi_s \sin\varphi_f + \cos\varphi_s \cos\varphi_f \cos(\lambda_f - \lambda_s)\right) \tag{5.1}$$

Parsing

In order to get forecast from Västtrafik, we must parse the XML that is returned by the provider. XML Parsing is made simple in ruby thanks to its Gems, so there is no much need to develop the subject. In our application, we used Nokogiri [Nok 10] to parse the data.

On the other hand, retrieving data from Storstockholms Lokaltrafik (SL) is less straightforward, and a large part of the parsing algorithm had to be implemented manually.

In order to parse the replies from SL's web services, we made intensive use of regular expressions. Regular Expressions (or Regex) are used to define a pattern in a string. In Ruby, Regex are Objects and can be declared as follows:

```
r = /[a-zA-ZöäåÖÄÅ]+/
```

This Regex describes a succession of one or more arbitrary letters from the Swedish alphabet, be they capital or minuscule. It has to be noted that UTF-8 encoding is enabled within regex in Ruby.

Once we have a regular expression, we can scan a string to find its sub-strings matching the Regex. Figure 5.9 gives an example of string parsing with the following pattern: ”Number␣Destination Name␣Number␣min”.

```
tram_regexp = /\d+\s[a-zA-ZöäåÖÄÅ\s\.]+\d+\smin/

text1 = "Next departure: Line 5 Lilla Torg. 25 min"
matches = text1.scan(tram_regexp) # => ["5 Lilla Torg. 25 min"]

text2 = "There is no next departure"
matches = text2.scan(tram_regexp) # => []
```

Figure 5.9.: Simple example of string parsing with Regular Expressions

5.3. Results

Everything works fine.
(give numbers)

6. Conclusion

6.1. On the AppStore

The first release of the application in its revision number 0.9 was uploaded under the name "Hållplats Väst"on the AppStore on March the 8^{th} and was ready for sale on March the 13^{th}.

After one month on the store, more than 400 units were downloaded.

6.2. Further Considerations

6.2.1. on the Client

Implement image recognition.

6.2.2. on the Server

Extend to more cities in Sweden or even Europe.

6.3. Summary

References

[Cau 92] Caudell, T.P. and Mizell, D.W: *Augmented reality: an application of heads-up display technology to manual manufacturing processes*, Res. & Technol., Boeing Comput. Services, Seattle, January 1992

[Sut 68] Sutherland, I. E. , *A head-mounted three dimensional display*, Proceedings of the December 9-11, 1968, fall joint computer conference, part I, San Francisco, California, December 1968

[Mil 94] Milgram, P., H. Takemura, et al., *Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum.* SPIE Proceedings: Telemanipulator and Telepresence Technologies . H. Das, SPIE. 2351 : 282-292, 1994

[Hay 09] Hayes, G., http://www.personalizemedia.com/16-top-augmented-reality-business-models/

[Erl 10] http://www.erlang.org/doc/reference_manual/

[Mne 10] http://www.erlang.org/doc/apps/mnesia/

[Fle 09] Fleckenstein, S. and Preston-Werner, T. : http://github.com/mojombo/erlectricity/

[Nok 10] Nokogiri

A. Ruby Code

File: bus_stop.rb

```ruby
class BusStop < ActiveRecord::Base
  validates_presence_of :name, :lat, :lng, :stop_id, :provider_name

  after_initialize :extend_module

  def url
    get_forecast_url(stop_id)
  end

  private
  def extend_module
    self.extend "Provider::#{provider_name}".constantize
  end

end
```

File: echo.rb

```ruby
# coding: utf-8

def get_stops(lat,lng)
  result = []
  stop_list = get_nearest_stops(lat,lng)
  stop_list.each do |element|
    attributes = []
    attributes << [:name, element.name]
    attributes << [:lat, element.lat]
    attributes << [:lng, element.lng]
    attributes << [:forecast, []]
    attributes << [:provider_name, element.provider_name]
    attributes << [:stop_id, element.stop_id]
    attributes << [:url, element.url]
    result << attributes
  end
  return result
end

def get_nearest_stops(lat,lng)
  lat = Float lat
  lng = Float lng
  area = 0.002
  list = BusStop.where(["lat > ? AND lat < ? AND lng > ? AND lng < ?",
                         lat - area, lat + area, lng- area, lng + area])
  while list.count < 10 do
    list = list + BusStop.where(
        ["lat > ? AND lat < ? AND lng > ? AND lng < ?
          AND NOT (lat > ? AND lat < ? AND lng > ? AND lng < ?)",
                                                     lat - 2*area,
                                                     lat + 2*area,
                                                      lng- 2*area,
                                                     lng + 2*area,
                                                       lat - area,
                                                       lat + area,
                                                        lng- area,
                                                       lng + area])
    area = area * 2
  end
  return sort_stops(list, lat, lng).first(10)
end

def sort_stops(list, lat, lng)
  #sort by distance to origin
  list.sort! do |a,b|
    distance_geodesic(Float(a.lat), Float(a.lng), lat, lng) <=>
      distance_geodesic(Float(b.lat), Float(b.lng), lat, lng)
  end
  return list
end

def distance_geodesic(lat1, long1, lat2, long2)
  #convert from degrees to radians
  a1 = lat1 * (Math::PI / 180)
  b1 = long1 * (Math::PI / 180)
  a2 = lat2 * (Math::PI / 180)
  b2 = long2 * (Math::PI / 180)

  r = 6356.75 #radius of earth
  #do the calculation with radians as units
  d = r * Math.acos(
          Math.cos(a1)*Math.cos(b1)*Math.cos(a2)*Math.cos(b2) +
          Math.cos(a1)*Math.sin(b1)*Math.cos(a2)*Math.sin(b2) +
          Math.sin(a1)*Math.sin(a2));

end

receive do |f|
  f.when([:parse, Array]) do |array|
    result = []
    i = 0
    while array[i] do
      element = {}
      stop_info = array[i][0]
      j = 0
      while stop_info[j] do
        element[stop_info[j][0].to_sym] = stop_info[j][1]
        j += 1
      end
      tobeparsed = array[i][1]
      loader = BusStopLoader.new(
                     "Provider::#{element[:provider_name]}".constantize)
      parsed = loader.parse_forecast(tobeparsed.force_encoding("UTF-8"),
                                                    element[:poi_id],
                                     element[:name].force_encoding("UTF-8"))
      element[:forecast] = parsed
      result << element
      i += 1
    end

    f.send!([:result, result.to_json])
    f.receive_loop
  end
  f.when([:echo, Array]) do |array|
    result = {}
    i = 0
    while array[i] do
      result[array[i][0]] = array[i][1].map(&:chr).join
      i += 1
    end
    nearest_stops = get_stops(result[:lat], result[:lng])
    f.send!([:result, nearest_stops])

    f.receive_loop
  end
end
```

File: provider.rb

```
module Provider

  module Base
    require 'timeout'
    require 'net/http'
    require 'uri'

    def delete_obsolete_stops(provider_name)
      BusStop.delete_all ["provider_name = ? AND updated_at < ?",
                                                    provider_name,
                                                     24.hours.ago]
    end
  end
end
```

File: vasttrafik.rb

```
module Provider::Vasttrafik
  include Provider::Base
  def update_stop_database
    provider_name = "Vasttrafik"
    use_identifier

    xml_tree = fetch_all_stops()
    return nil unless xml_tree
    rt90grid = SwedishGrid.new(:rt90)

    xml_tree.css("all_stops item").each do |element|
      stop_id = element.attribute("stop_id").try(:text).to_s
      bus_stop =
        BusStop.find_or_initialize_by_stop_id_and_provider_name(stop_id,
                                                                provider_name)

      rt90_x = Float(element.attribute("rt90_x").try(:text).to_s)
      rt90_y = Float(element.attribute("rt90_y").try(:text).to_s)
      latlng = rt90grid.grid_to_geodetic(rt90_x, rt90_y)

      bus_stop.provider_name = provider_name
      bus_stop.name          = element.css("stop_name").try(:text).to_s
      bus_stop.lat           = latlng[0]
      bus_stop.lng           = latlng[1]
      bus_stop.touch # ?
      bus_stop.save
    end

    delete_obsolete_stops(provider_name)
  end

  def get_forecast_url(poi_id)
    use_identifier
    url =
      "http://www.vasttrafik.se/External_Services/NextTrip.asmx/" +
                                          "GetForecast?identifier=" +
                                                        @identifier +
                                                         "&stopId=" +
                                                             poi_id
  end

  def parse_forecast(response, poi_id, poi_name)
    provider_name = "Vasttrafik"

    xml_tree = Nokogiri::XML.parse(response, nil, 'UTF-8')
    return [] if xml_tree.css("string").size == 0
    encapsuled_xml = xml_tree.css("string").text
    xml_tree = Nokogiri::XML.parse(encapsuled_xml, nil, 'UTF-8')
    # Extract data
    lines = []
    xml_tree.css("forecast item").each do |element|
      attributes = {}
      attributes[:line_number]           =
        element.attribute("line_number").try(:text).to_s
      attributes[:color]                 =
        element.attribute("line_number_foreground_color").try(:text).to_s
      attributes[:background_color]      =
        element.attribute("line_number_background_color").try(:text).to_s
      attributes[:destination]           =
        element.css("destination").try(:text).to_s

      attributes[:next_trip]             =
        element.attribute("next_trip").try(:text).to_s
      alternate_text                     =
        element.attribute("next_trip_alternate_text").try(:text).to_s
      attributes[:next_handicap]         =
        (alternate_text == "Handikappanpassad".force_encoding("UTF-8"))
      attributes[:next_low_floor]        =
        (alternate_text == "Low Floor")

      attributes[:next_next_trip]        =
        element.attribute("next_next_trip").try(:text).to_s
      alternate_text                     =
       element.attribute("next_next_trip_alternate_text").try(:text).to_s
      attributes[:next_next_handicap]    =
        (alternate_text == "Handikappanpassad")
      attributes[:next_next_low_floor]   =
        (alternate_text == "Low Floor")

      lines << [attributes[:line_number], attributes]
    end

    return sort_lines(lines)
  end

  private
  def fetch_all_stops
```

```
    get_poi_list_url = "http://www.vasttrafik.se/External_Services/" +
                        "TravelPlanner.asmx/GetAllStops?identifier=" +
                                                                @identifier
    xml_tree = fetch_xml_from_url_string(get_poi_list_url)
  end

  def fetch_forecast(poi_id)
    get_poi_forecast_url =
                        "http://www.vasttrafik.se/External_Services/" +
                              "NextTrip.asmx/GetForecast?identifier=" +
                                                            @identifier +
                                                             "&stopId=" +
                                                                poi_id
    xml_tree = fetch_xml_from_url_string(get_poi_forecast_url)
  end
end
```

File: storstockholms_lokaltrafik.rb

```
# coding: utf-8
module Provider::StorstockholmsLokaltrafik
  include Provider::Base

  def update_stop_database
    provider_name = "StorstockholmsLokaltrafik"
    for stop_id in 0000..9999 do
      a_stop = fetch_stop_with_id("%04d" % stop_id)
      if not a_stop == nil
        test_valid = get_forecast("%04d" % stop_id, a_stop[:name]) != []
        if test_valid
          bus_stop =
            BusStop.find_or_initialize_by_stop_id_and_provider_name(
                                                                  a_stop,
                                                           provider_name)
          bus_stop.provider_name = provider_name
          bus_stop.name          = a_stop[:name]
          bus_stop.lat           = a_stop[:lat]
          bus_stop.lng           = a_stop[:lng]
          bus_stop.touch # ?
          bus_stop.save
        end
      end
    end
    delete_obsolete_stops(provider_name)
  end

  def get_forecast_url(poi_id)
    url = "http://realtid.sl.se/?epslanguage=SV&WbSgnMdl=" +
                                                    poi_id +
                                          "-U3Rv-_-1-_-1-1"
  end

  def parse_forecast(response, poi_id, poi_name)
    html_tree = Nokogiri::HTML(response)

    lines = []

    # find Bus Forecasts
    bus_forecast = html_tree.xpath('//div[@class="TrafficTypeItem"]')
    bus_forecast.each do |element|
      stop_name = element.css('div h3').text.split(", ")[1]
      if(stop_name == poi_name)
        departure_list =    element.css('div[@class="Departure"]') +
          element.css('div[@class="DepartureAlternating"]')
        departure_list.each do |departure|
          next_trip_text = departure.css('span[@class="DisplayTime"]').
            text.squeeze(" ").strip
          if next_trip_text =~ /[Nn]u/
            next_trip = "0"
          else
            next_trip = departure.css('span[@class="DisplayTime"]').
              text.squeeze(" ").strip.scan(/\d+\smin/).first
          end
          if next_trip != nil
            attributes = {}
            attributes[:line_number]       =
              departure.css('span[@class="LineDesignation"]').
                text.squeeze(" ").strip
            attributes[:color]             = "#FFFFFF"
            attributes[:background_color]  = "#BB0000"
            attributes[:destination]       =
              departure.css('span[@class="DestinationName"]').
                text.squeeze(" ").strip
            attributes[:next_trip]         = next_trip.scan(/\d+/).first
            attributes[:next_handicap]        = false
            attributes[:next_low_floor]       = false
            attributes[:next_next_trip]       = ""
            attributes[:next_next_handicap]   = false
            attributes[:next_next_low_floor]  = false

            lines << [attributes[:line_number], attributes]
          end
        end
      end
    end

    # find Tram Forecasts
    tram_forecast = html_tree.xpath('//div[@class="TPIHolder"]')
    is_correct_stop = false
    tram_forecast.css('span[@class="Row1"]').each do |element|
      is_correct_stop |= (element.text == poi_name)
    end
    if is_correct_stop
      colors = {"T10" => "#0080FF",
                "T11" => "#0080FF",
                "T13" => "#FF0000",
                "T14" => "#FF0000",
                "T17" => "#00BB00",
                "T18" => "#00BB00",
                "T19" => "#00BB00"}

      departures = tram_forecast.css('div[@class="Row2"]')
      tram_regexp = /\d+[a-zA-ZöäåÖÄÅ\s\.]+\d+\smin/

      list_tmp = []
      departures.each do |departure|
        matches = departure.text.scan(tram_regexp)
        if matches != nil
          matches.each do |match|
            list_tmp << match
          end
        end
      end
      list_tmp.each do |to_parse|
        parsed = to_parse.squeeze(" ").split(" ")
        parsed_size = parsed.count
        line_number = "T" + parsed[0]
        next_trip = parsed[parsed_size-2]
        destination = parsed[1]
        for i in 2..parsed_size-3
          destination += " " + parsed[i]
        end

        attributes = {}
        attributes[:line_number]         = line_number
        attributes[:color]               = "#FFFFFF"
        attributes[:background_color]    = colors[line_number]
        attributes[:destination]         = destination
        attributes[:next_trip]           = next_trip
        attributes[:next_handicap]          = false
        attributes[:next_low_floor]         = false
        attributes[:next_next_trip]      = ""
        attributes[:next_next_handicap]          = false
        attributes[:next_next_low_floor]         = false
        lines << [attributes[:line_number], attributes]
      end
    end

    # find Pendeltåg
    pendeltag_forecast = html_tree.xpath('//div[@class="PPIHolder"]')
    pendeltag_header =
      pendeltag_forecast.css('div[@class="Header"]').css('h3')

    is_correct_stop = (pendeltag_header.text =~ /#{poi_name}/)
    if is_correct_stop
      Time.zone = "Stockholm"
      current_hour    = Time.zone.now.hour
      current_minute  = Time.zone.now.min
      departure_list  =
            pendeltag_forecast.css('div[@class="Departure TrainRow"]') +
            pendeltag_forecast.
        css('div[@class="DepartureAlternating TrainRow"]')
departure_list.each do |departure|
  striped_name = departure.css('span[@class="TrainCell Col2"]').
    text.scan(/[a-zA-ZöäåÖÄÅ\s]+/)
  destination = striped_name[0]
  for i in 1..striped_name.count - 2
    destination += " " + striped_name[i]
  end

  unformated_time = departure.css('span[@class="TrainCell"]').
    text.scan(/\d\d*:\d\d/)[0]
  if unformated_time == nil
    unformated_time =
      departure.css('span[@class="TrainCell Col1"]').
        text.scan(/\d\d*:\d\d/)[0]
  end
  splited_time = unformated_time.split(":")
  departure_hour = splited_time[0].to_i
  departure_minute = splited_time[1].to_i

  time_num = 60*(departure_hour - current_hour) +
              (departure_minute - current_minute)
  if time_num >=0
    next_trip = time_num.to_s
  else
    next_trip = ""
  end
  attributes = {}
  attributes[:line_number]       = "Pendeltåg"
  attributes[:color]             = "#FFFFFF"
  attributes[:background_color]  = "#000000"
  attributes[:destination]       = destination
  attributes[:next_trip]         = next_trip
  attributes[:next_handicap]        = false
  attributes[:next_low_floor]       = false
  attributes[:next_next_trip]    = ""
  attributes[:next_next_handicap]        = false
  attributes[:next_next_low_floor]       = false
  lines << [attributes[:line_number], attributes]
end
```

```
    end

    return sort_lines(lines)
  end

  private

  def fetch_stop_with_id(id)
    params = {
      "REQ0HafasSearchForw" =>"1",
      "REQ0JourneyDate"     =>"25.03.10",
      "REQ0JourneyStopsS0A" =>"255",
      "REQ0JourneyStopsS0G" =>id,
      "REQ0JourneyStopsSID" =>"",
      "REQ0JourneyStopsZ0A" =>"255",
      "REQ0JourneyStopsZ0G" =>"",
      "REQ0JourneyStopsZID" =>"",
      "REQ0JourneyTime"     =>"10:10",
      "existUnsharpSearch"  =>"yes",
      "ignoreTypeCheck"     =>"yes",
      "queryPageDisplayed"  =>"no",
      "start"               =>"Sök resa",
      "start.x"             =>"0",
      "start.y"             =>"0"
    }

    x = Net::HTTP.post_form(URI.parse(
      "http://reseplanerare.sl.se/bin/query.exe/sn"), params)
    html_tree = Nokogiri::HTML(x.body)

    error = html_tree.xpath('//label[@class="ErrorText"]')

    return nil unless error.empty?

    form = html_tree.xpath('//div[@class="FieldRow"]').first
    name = form.css("strong").text

    latlong_unparsed =
      form.xpath('//input[@type="submit"]').attribute("name").value
    lat = nil
    lng = nil
    latlong_unparsed.split('&').each do |element|
      attribute = element.split('=')
      if(attribute[0] == "REQMapRoute0.Location0.Y")
        lat = Float(attribute[1])
      end
      if(attribute[0] == "REQMapRoute0.Location0.X")
        lng = Float(attribute[1])
      end
    end

    attributes = {}
    attributes[:name] = name.split(" (")[0]
    attributes[:lat] = lat / 1000000
    attributes[:lng] = lng / 1000000
    attributes[:stop_id] = id

    return attributes
  end

end
```

B. Erlang Code

File: request_handler.erl

```
-module(request_handler).
-export([start/0, restart_driver/0, handle_request/1]).

start() ->
  inets:start(),
  cache_server:start(),
  spawn(fun() -> start_driver() end).

start_driver() ->
  io:format("New Port opened from ~p~n", [self()]),
  Cmd = "rails runner ./lib/echo.rb",
  Port = open_port({spawn, Cmd}, [{packet, 4}, nouse_stdio, exit_status,
    binary]),
  register(ruby_port, Port),
  DriverPid = spawn(fun() -> ruby_driver(Port) end),
  port_connect(Port, DriverPid),
  register(ruby_driver, DriverPid),
  ok.

restart_driver() ->
  exit(whereis(ruby_driver), kill),
  port_close(whereis(ruby_port)),
  start().

ruby_driver(Port) ->
  port_connect(Port, self()),
  link(Port),
  receive
    {transmit, Payload, FromPid} ->
      FromPid!{ok, driver_call(Port, Payload)},
      ruby_driver(Port);
    {Port,{exit_status,_}} ->
```

```
      start();
    {'EXIT', Port, Reason} ->
      io:format("~p ~n", [Reason]),
      start()
  end.

driver_call(Port, Payload) ->
  port_command(Port, Payload),
  receive
      {Port, {data, Data}} ->
        {result, _Text} = binary_to_term(Data)
  end.

handle_request(Params) ->
  MyPid = self(),
  Payload = term_to_binary({echo, Params}),
  ruby_driver!{transmit, Payload, self()},
  receive
    {ok, {result, StopList}} ->
    spawn(fun() -> fetch_forecasts(StopList, MyPid) end),
    receive
      {ok, {forecast, ForecastList}} ->
        %io_lib:format("~p~n", [ForecastList])
        ToParse = term_to_binary({parse, ForecastList}),
        ruby_driver!{transmit, ToParse, MyPid},
        receive
          {ok, {result, FinalResult}} ->
            FinalResult
        end
    end
  end.

fetch_forecasts(StopList, ToPid) ->
  MyPid = self(),
  [spawn(fun() -> fetch_forecast(Stop, MyPid) end) ||
    Stop <- tuple_to_list(StopList)],
  ForecastList = wait_for_forecasts([]),
  ToPid!{ok,{forecast, ForecastList}}.

wait_for_forecasts(ForecastList) ->
```

```
  receive
    {ok, Forecast} ->
      if
        length(ForecastList) < 9 ->
          wait_for_forecasts([Forecast | ForecastList]);
        true ->
          _Result = [Forecast | ForecastList]
      end
  after
    5000 ->
      _Result = ForecastList
  end.

fetch_forecast(Stop, ToPid) ->
  StopBundle = tuple_to_list(Stop),
  [URL] = [binary_to_list(Value) ||
    {Atom, Value} <- StopBundle, Atom == url],
  Body = cache_server:get(URL),
  ToPid!{ok, {StopBundle, Body}}.
```

File: cache_server.erl

```
-module(cache_server).
-export([start/0, get/1, cleaner_daemon/0]).
-record(cached_url, {url, timestamp, data}).

start() ->
  mnesia:start(),
  mnesia:create_table(cached_url, [{attributes,
      record_info(fields, cached_url)}]),
  spawn(fun() -> cleaner_daemon() end).

cleaner_daemon() ->
  F = fun() ->
      {Mega, Sec, _Micro} = erlang:now(),
      T = Mega * 1000000 + Sec - 600,
      TimeStamp = {T div 1000000, T rem 1000000, 0},
      MatchHead = #cached_url{url = '$1', timestamp = '$2', data = '_'},
      io:format("~p~n", [TimeStamp]),
      Guard = {'<', '$2', {TimeStamp}},
      Result = '$1',
      ToDelete = mnesia:select(cached_url,
            [{MatchHead, [Guard], [Result]}]),
      [ mnesia:delete({cached_url, Url}) || Url<-ToDelete]
    end,
  receive
  after
    600000 ->
      mnesia:transaction(F),
      cleaner_daemon()
  end.

get(Url) ->
  case request_mnesia(Url) of
    {atomic,[[TimeStamp,DataTmp]]} ->
      case is_timestamp_valid(TimeStamp) of
        true ->
          Data = DataTmp;
        false ->
          Data = get_over_http(Url),
          spawn(fun() -> write_mnesia(Url, Data) end)
      end;
    _ ->
      Data = get_over_http(Url),
      spawn(fun() -> write_mnesia(Url, Data) end)
  end,
  Data.

get_over_http(Url) ->
  {ok, RequestId} = http:request(get, {Url, []}, [], [{sync, false}]),
  receive
    {http, {RequestId, Result}} ->
      case Result of
        {{"HTTP/1.1",200,"OK"}, Header, Body} ->
          UglyTest = [ 'DIEVASTTRAFIKDIE'||
              {"content-length", "2442"} <- Header],
          case UglyTest of
            [] ->
              Body;
            _ ->
```

```
              get_over_http(Url)
          end;
        _ ->
          get_over_http(Url)
      end
  after
    3000 ->
      get_over_http(Url)
  end.

is_timestamp_valid({MegaSec, Sec, _MicroSec}) ->
  TimeStamp = MegaSec*1000000+Sec,
  {NowMega, NowSec, _NowMicro} = erlang:now(),
  NowStamp = NowMega*1000000+NowSec,
  NowStamp - TimeStamp < 45.

request_mnesia(Url) ->
  F = fun() ->
      Cache = #cached_url{url = Url, timestamp = '$1', data = '$2'},
      mnesia:select(cached_url, [{Cache, [], [['$1', '$2']]}])
    end,
  mnesia:transaction(F).

write_mnesia(Url, Data) ->
  Cache  = #cached_url{  url= Url,
              timestamp = erlang:now(),
                         data = Data},
  F = fun() ->
      mnesia:write(Cache)
    end,
  mnesia:transaction(F).
```