GetSound

A noise level storage and visualization system using models to generate probabilistic data

Master Thesis

Mariusz Wisniewski

September 2012

Thesis supervisors:

Prof. Philippe Cudre-Mauroux
eXascale Infolab

UNIVERSITÉ DE FRIBOURG SUISSE
UNIVERSITÄT FREIBURG SCHWEIZ

eXascale Infolab
Department of Informatics
University of Fribourg (Switzerland)

eXascale Infolab

Acknowledgements

I want to thank the people who have worked with me on this project and helped me in the various stages of development of this system. My supervisor *Prof. Philippe Cudre-Mauroux* who has given me the opportunity and the means to do this project and guided me through its whole process and whose advices have been very valuable. He has always been supportive and gave me the motivation to push things forward. His contribution to the modelling part of the system has been very valuable.

My colleague *Jean-Gérard Pont* of the *eXascale Infolab* who helped me in the implementation of the visualization part of the project and whose advices during the development of the project have been very valuable. His knowledge of *dipLODocus* has helped me to solve many problems.

My colleague *Marcin Wylot* of the *eXascale Infolab* who helped me setting up the development environment of the project and whose many advices have been very valuable. He helped me a lot during the integration of *dipLODocus* in the system and always found time to explain its functionalities to me.

Dr. Apostolos Malatras, whose interest in the project and advices helped me in the implementation of the modelling part of the system.

My friend *Patrick Bruttin*, who helped me during the phase of the project. He helped me in taking all the measures necessary to analyze the correctness of my models.

My fellow student *Iliya Enchev* who helped me setting up TeXclipse and gave me the template used to write this report.

The whole *eXascale Infolab* at the University of Fribourg headed by *Prof. Dr. Philippe Cudre-Mauroux* who have been very supportive. They have always found the time to help me and to give me counsel. Thanks to *Gianluca Demartini, Djellel Difallah, Roman Prokofyev, Alberto Tonon, Michael Luggen.*

I would also like to express my gratitude to my family and friends who have encouraged me during the development of the project.

Abstract

We think that a system allowing its users to visualize the noise level of any region at any time would be really useful in many cases. For example, when looking for an apartment in a quiet neighborhood.

Many systems allowing to visualize the noise levels around roads and freeways already exist in Switzerland. Some cities (like Geneva) have sensors in some areas of interest and use them to record the noise levels. Then, the results of these measurements can be visualized on a web page. [6]
Our idea is to reproduce such a system but we want to improve it by covering bigger regions like a country, and we want to be able to have continuous measurements taken everywhere at every moment in order to improve the options available for our users during the visualization.

To do so, we need to be able to store a huge amount of data that can be accessed and read very quickly. We also need to be able to cover the region in which we want to measure the noise levels in its whole. We also want to be able to provide data when no measurement has been made for a given place at a given time.

Finally, we want our system to provide an efficient way for the user to visualize the data and to give him different options that can improve the visualization.

GetSound is such a system and offers all these features. We will describe it in details throughout this report.

Keywords: *noise, coordinates, dipLODocus, database management system, RDF, triples, models, probabilistic data, interpolation, web service, application, smartphone, crowd-sourcing, GPS, PHP, visualization.*

Table of Contents

List of Figures

Listings

1 Introduction

The main idea of this project is for the users to have a system allowing them to visualize the noise levels of a given region (for example Switzerland).
Such a system could be useful in many cases, such as looking for a quiet neighborhood to buy a house or an apartment.

We want to be able to cover a huge area in which as many measurements regarding noise levels as possible would be made at every moment. This is obviously an ambitious goal and nearly impossible to achieve unless we put sensors all over Switzerland and engage hundreds of technicians. However, we think good results can be achieved using crowdsourcing. Indeed, almost everyone nowadays has a smartphone with a GPS and a microphone which are the main tools we need to measure noise levels and to correlate them with an area. Therefore we decided it would be best to build a smartphone application that will serve as a sensor, measuring noise and gathering values and coordinates.

The data collected in this manner by our application will need to be stored in a database. Therefore, our system will need to include a database management system that will be able to deal with the needs of the other parts of the system and the huge amount of data that we want to store.

In order to have a maximum amount of data, our system will also need to create probabilistic data out of the data measured by using various models. Indeed, unless a huge amount of users populate our database night and day, there will always be a case where our database does not contain any measures for a given place at a given time. Still, in some cases, we can create a probabilistic measure out of other measures that have been made.

The visualization part will be accessible through a website. Users will be able to visualize noise levels on a map and to apply different kinds of filters by modifying parameters such as time. Graphs will also be available in order for the users to get average values over a period of time and to see the evolution of the noise levels over time for a given place.

Finally, to coordinate all of these parts, we decided to implement a web service that will be the central part of the system. It will be responsible for receiving and parsing the

data received from the application, for creating the probabilistic data and for querying the database management system.

1.1. Motivation and Goals

The eXascale Infolab is actually developing *dipLODocus*; a database management system for RDF data processing. One of the motivation of doing this project was to have a system that is using *dipLODocus* and taking advantage of the possibilities it offers. We wanted to implement new functionalities for *dipLODocus* and have a visualization system calling and displaying data it contains.

Another goal was to find ways to create data out of data we receive using *models*. Having such *probabilistic data* is interesting because it gives us additional information which we would not have and creates various kinds of data with different relevance. It seemed challenging to build such a system and to implement its behaviour in regard of the *raw data*, *models* and the *probabilistic data*.

To summarize, the goals of the project are:

- to have a working system allowing to take measure and visualize the results on a map.
- to make use of *dipLODocus* and to enhance it by adding new functions, such as the visualization of the data and the remote querying of the database management system.
- to build *models* and to verify if the generated *probabilistic data* are representative of the reality.

1.2. Organization

The rest of the work is organised as follows: in the second chapter we will talk about the global architecture of the system without going in too much detail. In the third chapter we are describing the different elements composing the system in detail as well as the functionalities of the system. In the fourth chapter we are describing the visualization part of the system. In the fifth chapter we are describing the tests we ran on our system and the results we got. And finally, in the sixth chapter we are describing possible future implementations.

1.3. Notations and Conventions

Formatting conventions: *italic* is used for emphasis, especially over keywords.

The present report is divided in Chapters. Chapters are broken down into Sections. Where necessary, sections are further broken down into Subsections, and Subsections

may contain some Paragraphs.

Figures, Tables and Listings are numbered inside a chapter. For example, a reference to Figure j of Chapter i will be noted *Figure i.j*.

RDF triples are displayed as follows:

```
<subject> <predicate> "literal" .
```

2
Architecture of the system

In this chapter we will describe the architecture of the system and explain the choices we have made. The different elements of the system will be mentioned but will be explained in detail in the following chapters. Here we will have a global vision of the system.

2.1. A modular system

We wanted the architecture of our system to be modular. This way we have different elements which we can build, test and operate separately. This makes things alot easier during the developing part of the project. We can also replace or get rid of one of the elements far more easily this way. This is the main reason why we have chosen a modular architecture. Indeed we will have to combine different kinds of technology, and besides *dipLODocus* is still in development, so we had to be sure we could change any part if something was not working.

Figure 2.1 represents the architecture of our system.

As we can see, we have a PHP *web service* awaiting requests from Iphones or any computer through a web browser. This way, we can send short messages, containing our request, data and parameters, in an easy way instead of sending large SPARQL queries over the network. The *web service* will understand the message and will be able to behave

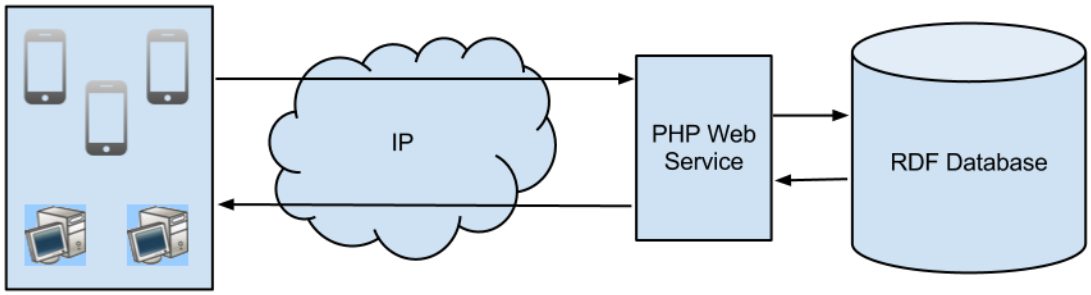

Figure 2.1.: The architecture of the system.

according to what is expected from it.

The *web service* will also deal with the management of the data; its transformation into RDF triples, the creation of *probabilistic data* using *models*, and the querying of the database. It is the heart, or rather the head of our system.

As we just mentioned, we will store our data as *RDF triples*. This will allow us to retrieve it much faster using *dipLODocus*, which will be important for the visualization part. RDF (Resource Description Framework) allows us to have a conceptual model of our data, which will allow us to extract huge amount of information out of our database really fast.

Concerning the database system that we use, we want to store our data using *dipLODocus*, a database system for RDF data management that is very fast for data retrieval and developed by the *eXascale Infolab*. However, *dipLODocus* being still in development, we had to use *Virtuoso*, another RDF database system, in the early stages of development of our system.

2.2. Choice of the smartphone

We have decided to use the IPhone for this project, since it is a very popular smartphone in Switzerland and it has a nice service development kit with lots of features, allowing users to build solid applications in an easy and efficient way.

All the development and testing have been done using an Iphone 4 with IOS 5.1 and 5.1.1 and XCode 4.4 and 4.4.1.

2.2.1. Problems encountered

In order to develop applications for Iphone, one has to own a Mac computer and buy a developer license. We had to purchase such a computer and license and I thank my supervisor *Prof. Philippe Cudre-Mauroux* for it.

2.3. Management of the data

The data we receive from our sensors contains a number of different variables. A measurement is made of coordinates, time and date, noise levels and probability values. Some of these values are measured by the sensor, and some are added afterwards by the web service.

2.3.1. Coordinates

The coordinates are usually composed of the latitude, longitude and altitude of a given place.

The latitude specifies the north-south position of a point on the Earth's surface. Lines of constant latitude, or parallels, run east-west as circles parallel to the equator. Latitude is an angle which ranges from 0° at the Equator to 90° (+90° North, -90° South) at the poles.
The longitude specifies the east-west position of a point on the Earth's surface. Longitude is an angle which ranges from 0° at Greenwich, England, to 180° (+180° East, -180° West).

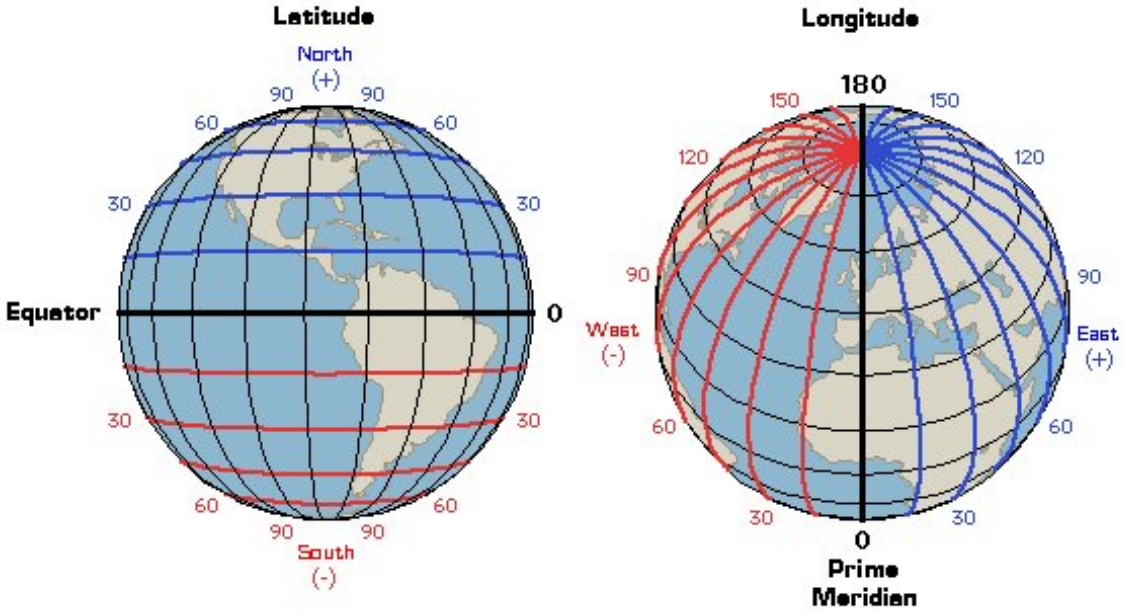

Figure 2.2.: Latitude and longitude. [7]

We chose to omit the altitude since we do not think it will be useful in our system. The coordinates are measured by the sensor (the smartphone in our case).

2.3.2. Time

In our system, time is composed of the date (year, month, day) and hour (hour, minute). We decided that the granularity of the time we take into account will be of 30 minutes. Indeed, we do not want to have unuseful data and think a more precise granularity will not improve the use of our system but, on the contrary, make things less comfortable for the user. Therefore, all measurements for 30 minutes will be averaged and processed before being inserted in the database. This way we will reduce the amount of triples stored in the database and accelerate the retrieval of data.

We also decided to add a timestamp to our measurements in order to accelerate the retrieval of data from the database. If we had to get a measurement from the database by using each element of the date and hour, the whole process would be slowed significantly. Indeed, the database system would have to make a lot of joins, comparisons and merges, which would take more time than finding a single value. We keep the date elements so that we have the possibility to query the database for measures regarding a given month or day for example, but we will mainly be using the timestamp as a time value.

Another element we decided to add is the day of the week. Indeed, some future queries will probably be about specific days. Therefore, instead of having to retrieve all the values and make calculations to get only a certain set of data we can directly make the query according to this value.

The time at which the measurement was made can be set by the web service in order to save bandwidth. The sensors have only to send the coordinates and values.

2.3.3. Noise level values

We are interested in two types of values; the average noise value and the peak noise value.

The average noise value represents the average of all the samples of a measure over a very short time (not specified in the documentation of the IOS SDK, but in the case of our application a noise level is measured every 0.1 second).

The peak noise value is the maximum value of these samples.

The noise level is measured in decibels. The decibel (dB) is a logarithmic unit that indicates the ratio of a physical quantity (usually power or intensity) relative to a specified or implied reference level. A ratio in decibels is ten times the logarithm to base 10 of the ratio of two power quantities.

The Iphone records audio over two channels. Therefore, we have four noise values per measurement.

2.3.4. Probability

Our system is building *probabilistic data* based on models each time new data is received from a sensor. This means that the data we store has to have some value indicating if it is raw data or *probabilistic data*, and in the latter case which model has been used to build it. This value will be checked by our models in order to know if better data already exists and if it is worth to proceed or not.

2.3.5. The RDF schema

In order to query the database and to retrieve the data we need, we have to have an RDF schema that we will follow when transforming the data into triples before its insertion in the database. For a definition of RDF go to chapter 3.3.1.

Figure 2.3 and Figure 2.4 represent our RDF schema and instance.

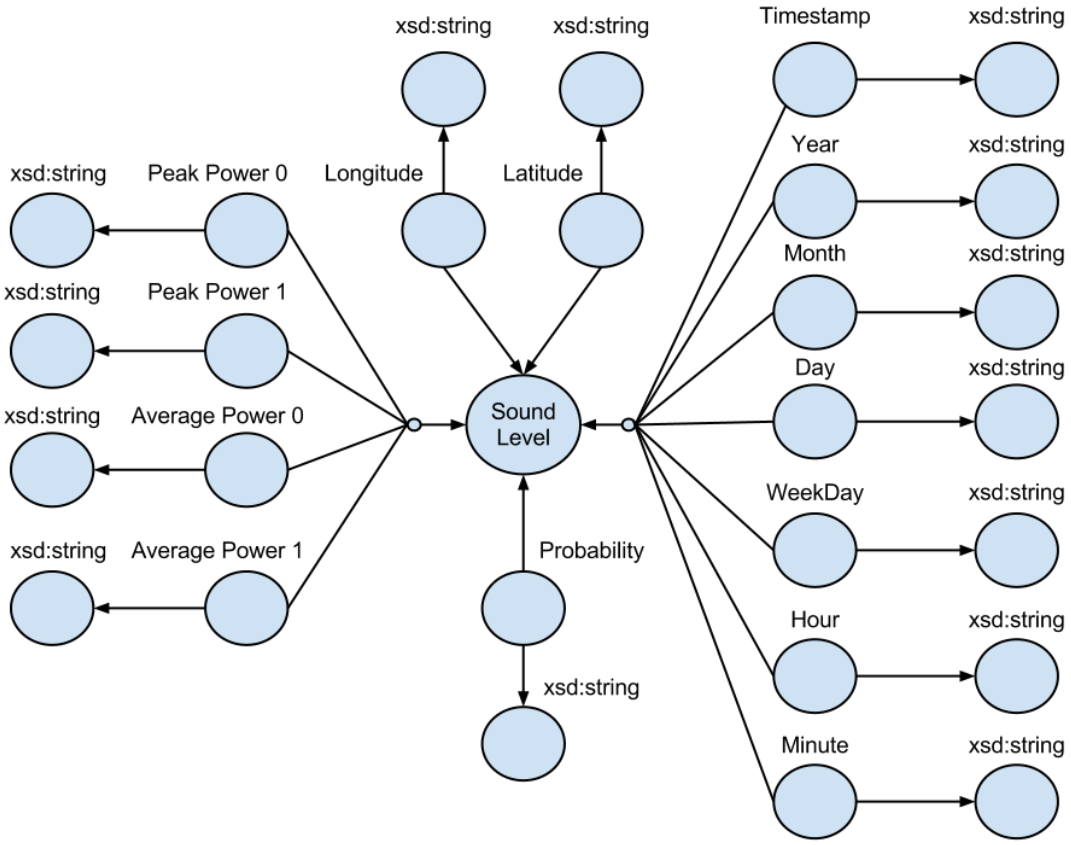

Figure 2.3.: The RDF schema of our data.

Listing 2.1 is an example of a set of rdf triple.

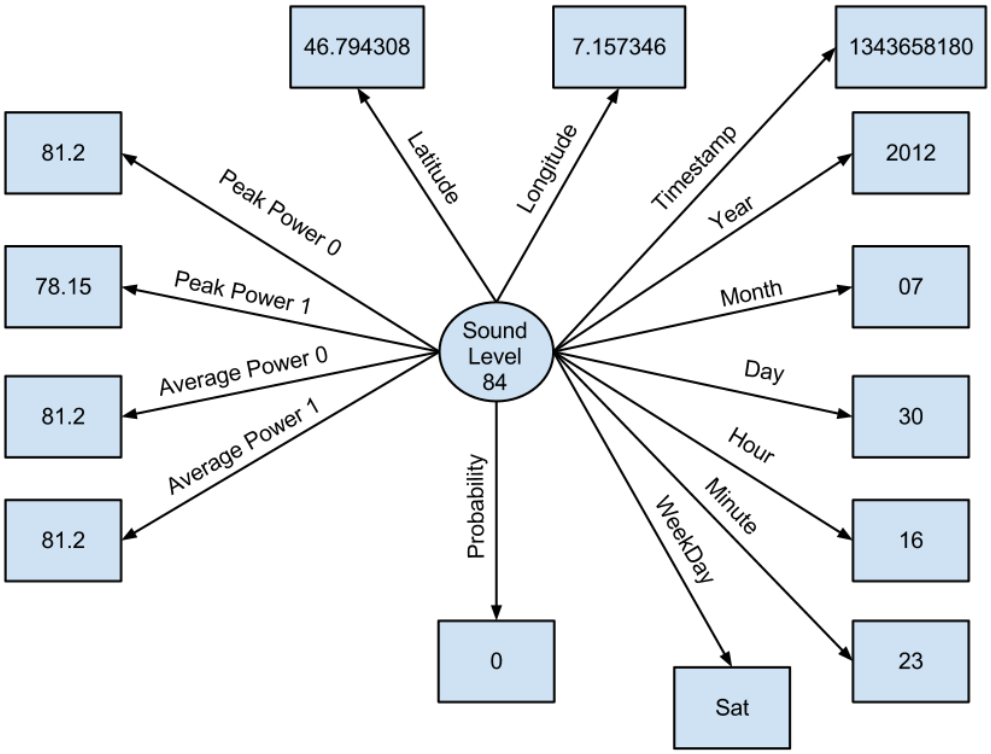

Figure 2.4.: An RDF instance of our data.

```
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#year> "2012" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#month> "07" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#day> "25" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#hour> "13" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#minute> "30" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#timestamp> "1343215800" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#latitude> "46.794263462963" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#longitude> "7.157410467741" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#probability> "0" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#averagePower0> "13.5" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#averagePower1> "13.6" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#peakPower0> "37.361" .
<http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#peakPower1> "37.761" .
```

Listing 2.1: A set of RDF triples.

We will see in the database part of the project that some additional triples have to be added to a set in order for *dipLODocus* to be able to process the data.

2.3.6. Probabilistic data and models

In our system, we want our users to be able to visualize the noise levels of a given region at a given time. However, our database will not always have measures for a given place at a precise moment. Still, there is always a possibility that we have other measurements

allowing us to deduce what could possibly be the values of the measure that is missing. Therefore, we need to create models [2] that will be used to build this *probabilistic data* whenever it is possible to. The best way would be to use interpolation.

Interpolation is a method of constructing new data points within the range of a discrete set of known data points. As soon as we receive a new measurement, we can check if we have other measurements in the database similar in terms of position or time. If we do, then we can deduce the values in between (as showed in Figure 2.5).

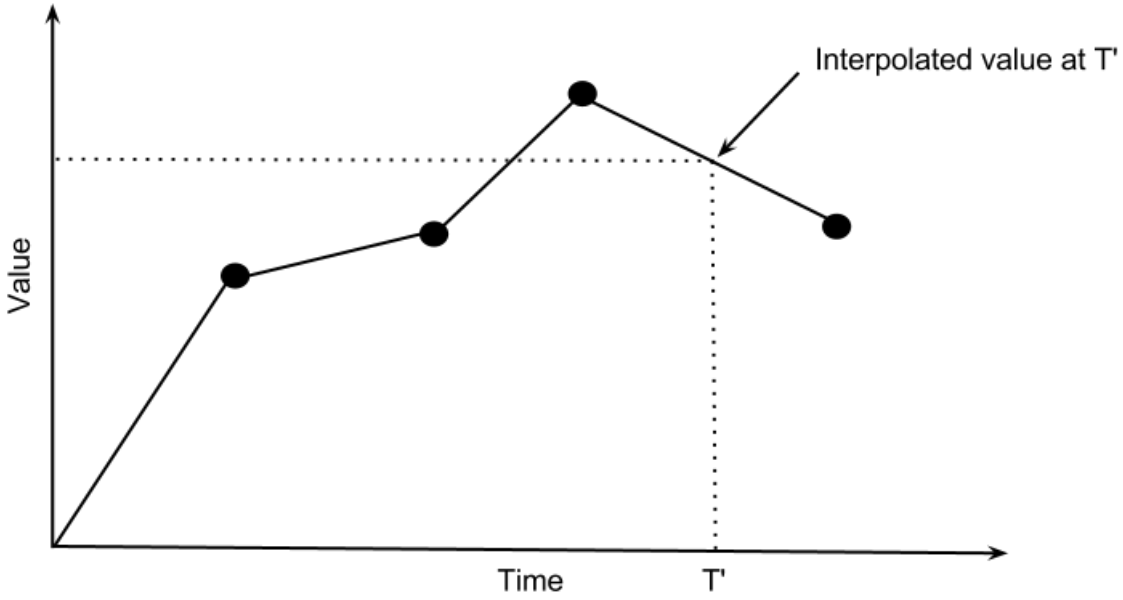

Figure 2.5.: An example of the interpolation of a value.

In this project we have two kind of models; spatial models and time models.

The spatial models are responsible for building *probabilistic data* in regard of distances. These models will interpolate the values of the measurement received by the web service and build data for the surrounding areas of the measurement. We have created two spatial models that we call depending on their probability level. This way less valuable data will not have to be generated if more valuable data already exists, which will make the whole process faster.

The time models are responsible for building data in regard of time and dates. These models will interpolate the values of the measurement received by the web service and build data for the same exact place but at different point in time. We have created two time models with different probability levels just as for the spatial models.

The different models are described in detail in chapter 3.2.2.

3

The elements of our system

We have described the architecture of our system in chapter 2. The system is modular and composed of many elements. In this chapter, we are going to describe each of these elements in detail. We will explain its use, its implementation and its deployment as well as the problems we encountered.

3.1. The IPhone application

The first purpose of the application is to measure values and send the data to the system. For this purpose, our IPhone application needs to have different elements.

First of all, we need to be able to record the sound. For this task, we use the AVFoundation framework which allows the developer to create recorders and players to manipulate audio.
We have added two buttons to the application, REC and PLAY which allow us to record sound, store it in a file, and play it. The user can stop the recording at any time by pressing the REC button again (which has the label STOP during the recording phase).

During the recording of the sound, the average and peak powers are measured every fixed amount of time using methods given by the AVAudioRecorder object and sent to the web service. We chose to take a measure every 0.1 second. These measures are stored in a

sorted array, and once we have 10 of them (after 1 second) we choose to keep the median value.
Therefore we really have one measurement per second and after a given number of seconds we send the highest average value per channel and the highest peak value per channel to the web service. We chose to do this every 10 seconds.

We then use the CoreLocation framework to get the GPS coordinates of the user with the best accuracy available.

All of these values, as well as the date and time, are displayed in real-time during the recording phase for checking and testing purpose.

We do not need to send the date and time to the web server since it will add these values to the measurement by itself.

Figure 3.1 shows a view of the application.

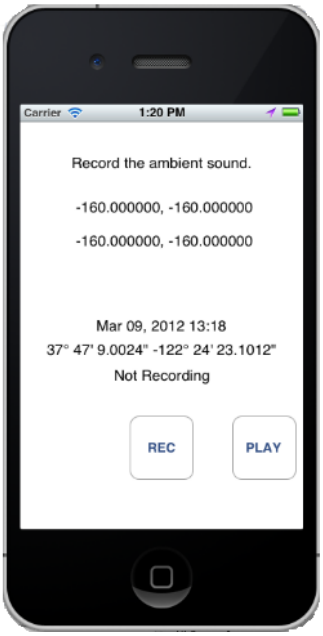

Figure 3.1.: A view of our application.

3.1.1. Problems encountered

The main problem with the Iphone is that the quality of its microphone does not allow us highly precise measures. It is limited to 100dB maximum (which is not so bad). However, the result obtained with our application are very similar to other decibelmeter applications such as Decibel Meter or Decibel Ultra.

Another point is the precision of its GPS that can be really bad indoors. However, our system is meant to make measures in the outside.

A major problem is that one cannot test his applications on a real IPhone unless he has bought a developer license. Without such a license, one has to test his application on a simulator integrated in the SDK, which is not as pragmatic.

We also had a problem after an upgrade of the IOS on the IPhone, where the application had to be recompiled using a newer version of the IOS SDK. Therefore, we had to upgrade our version of XCode, but add many problems during the process and lost quite some time.

The last point, which is not really a problem in itself, is that IPhone applications are coded using Objective C. Therefore, it was not easy starting to code.

3.2. The web service

The web service is the heart and head of our system. It is the medium between the other elements. It receives the measurements from the IPhone application, stores them temporarily before transforming them into *RDF triples* and sending them to the database system to store them permanently, contains the *models* it uses to generate the *probabilistic data* and contains all the logic fo the visualization. All the data, received or sent by the system passes through the web service and it manages all the system.

3.2.1. Reasons for having a web service

Modularity

We decided to build a web service for many reasons. First, we wanted to have a modular architecture. Even if we could implement all the behaviour contained in the web service in *dipLODocus*, we would run the risk that if something went wrong and we had to change the database management system, we would have to redo all the work. *dipLODocus* being still in development, we could not allow ourselves to take the risk.
Besides, such an approach is helpful during the development phase, since the architecture is clear and coding becomes easier. It also eases the workload of the other elements by taking care of some of the tasks that have to be performed.

Easy remote access

Having a PHP web service also gives us a simple way to invoke a given PHP file, to pass it parameters and to receive results. This way we can implement different behaviours by creating different PHP files, or calling different functions contained in the same file by adding different parameters in the URL.

On the IPhone application, we just have to build a socket and send an URL containing the parameters of our request. This saves bandwidth since we do not have to send huge

SPARQL queries. Any program or application can have access to our web service through a simple socket.
For the visualization part, any computer having an access to the internet can access our web service through a browser using an URL containing the parameters of the request. Here is an example of such an URL :

http://diuflx01/GetSound.php?query=insert&lat=46.794308&lng=7.157346&ap0=81.2&ap1=81.2&pp0=88.9&pp1=87.4

dipLODocus is also easily accessible from a PHP server through sockets. This feature was implemented by *Marcin Wylot* and is very useful for our project. We will describe it in chapter 3.3.2

Taking care of some of the system's workload

The web service will also deal with the modelling part. We think that since the transformation of the data into RDF triples has to be done after the creation of the *probabilistic data*, both steps should be done by the web service. Again, the modelling should be independent of what database management system we are using. This step can be done using PHP.
The web service and the database management service can be located on different machines. In this way and in case of high activity of the system, the workload would be divided between both elements.

Concurrency and security issues

By having only one way to remotely access the system, via the web service, security measures can be set up far more easily and concurrency can also be taken care of in a better way. We did not concentrate on these aspects though and leave them for future work on the project.

3.2.2. Behaviour of the web service

The Apache server is waiting for a request. Once someone connects, it gets the name of the PHP file and all the variables from the URL and parses them to understand what type of request it is asked to perform.

In the case of insertion of a new measurement, all the variables are retrieved and we check that the coordinates of the measurement is contained within the region we are covering. The covered region is defined by four constant values indicating the north and south latitudes and east and west longitudes (Figure 3.2).

Once we know the measurement coordinates are good we can add it to our temporary array.

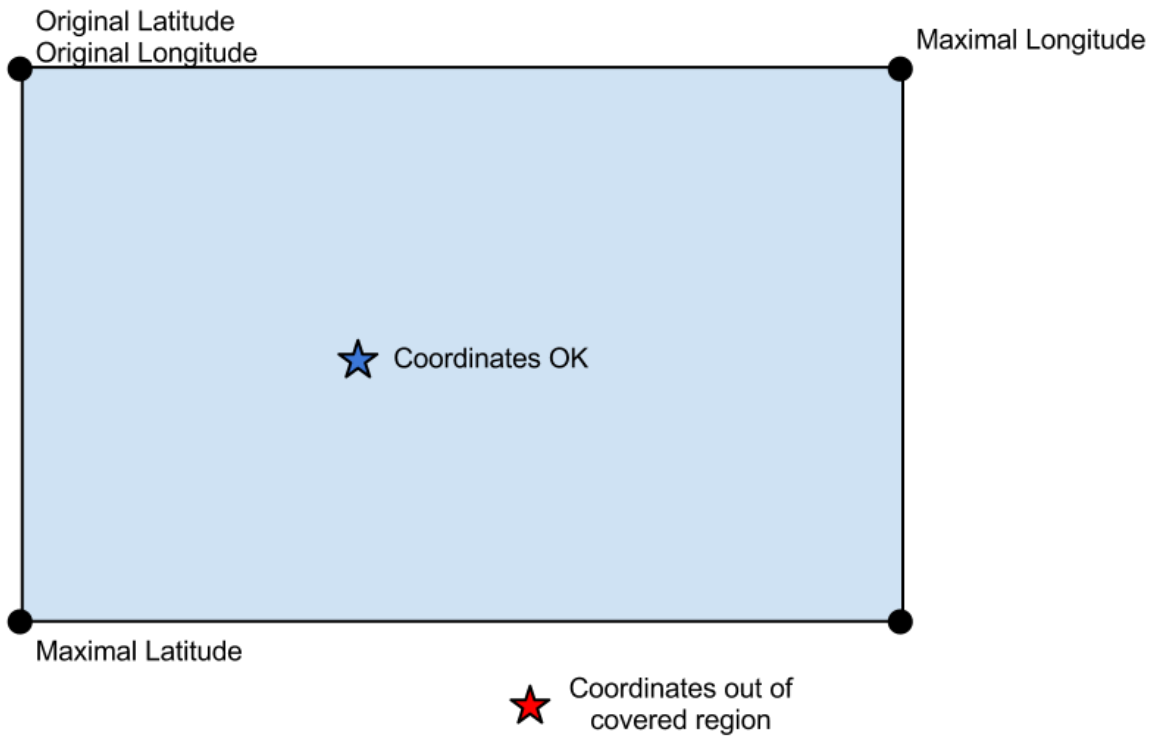

Figure 3.2.: Coordinates verification.

The temporary array

The purpose of the temporary array is to store all the measurements of the last 30 minutes as well as the *probabilistic data* built using our models, so that we can reach a time granularity of 30 minutes in our system. Once every 30 minutes we will fill the database with the content of the temporary array and empty the latter.
We also need a space granularity. We decided to divide the region we cover into 10 meters by 10 meters squares. Each element of the array corresponds to a square and holds its noise values. The latitude and longitude of each square are the latitude and longitude of its center, therefore if we measure some noise with coordinates corresponding to the edge of a square, the resulting data will correspond to the whole square with the center as its coordinates and a time granularity of 30 minutes. Obviously, when we write data for a given square, we do not need to write the coordinates, since we can deduce them from which element of the array we are working with.

The temporary array is a file stored on the server. We decided not to keep all the values in main memory in case the amount of data would exceed the available memory.
Each square is represented as a given number of characters so that we can reach the position we need very fast. This has a great disadvantage though, the file can use a lot of space even if we have few measures. Indeed the empty spaces between two measures are filled with useless data which still consumes space on the disk. However, we are more interested in the aspects of data access speed rather than disk space utilization.

Using the four constants indicating the limits of the area our system covers and the size

of a square, we know how many squares our array holds, and using the coordinates of a square we can access the array at the right position very fast.

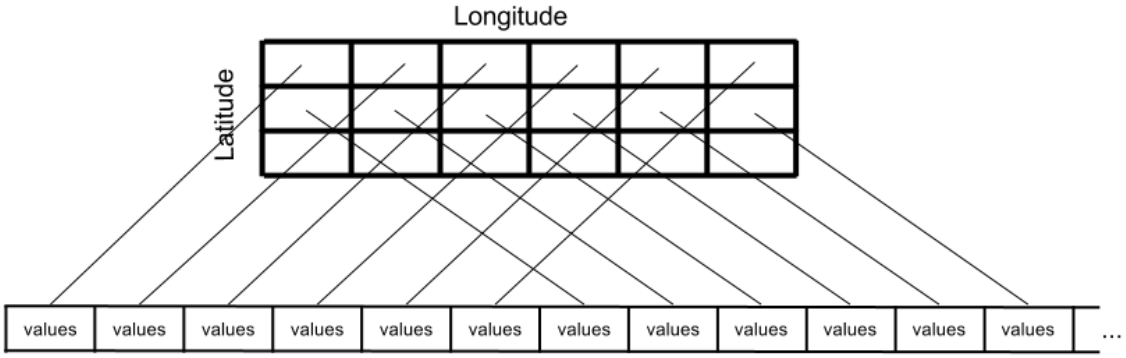

Figure 3.3.: The temporary array.

So once we have our measurement, we need to check in the array if another measurement already exists for this particular square. This is possible since the elements of the array each hold data about up to 30 minutes. If no data is already present, we add our values to the element of the array. If another measurement already exists, we check its probability value to know if it is raw data or *probabilistic data* built by a model. If it is *probabilistic data*, we can just overwrite it, since raw data is more accurate than *probabilistic data*. Otherwise we make an average of the all the measurements. This means that each square must hold a value indicating how many measurements have been used to make an average, so that we can reevaluate the value according to the whole number of measurements we have had up to now.

Once this has been done we can call the spatial models.

Spatial models

The first model called "the distance model" will check if another measurement already exist in a nearby square of the temporary array. If so, we will fill the gaps between the two measurements. We check every square up to a distance of 50 meters (being careful not to go beyond the array limits). Of course we do not check the direct neighbors of our measurement square since there is no gap to fill (this would be a waste of time).
Once another nearby measurement has been discovered we measure the distance between the two squares. This will be used to set the probability of the *probabilistic data* we are about to create. The closer the two measurements are, the more valuable the *probabilistic data* will be, and it will eventually overwrite previous less valuable *probabilistic data*.
The gap we fill is defined by the rectangle between our two measurements. The *probabilistic data* is created by interpolating the two measurements according to the number of steps we have between the two squares.

You can see the whole process in Figure 3.4, Figure 3.5 and Figure 3.6.

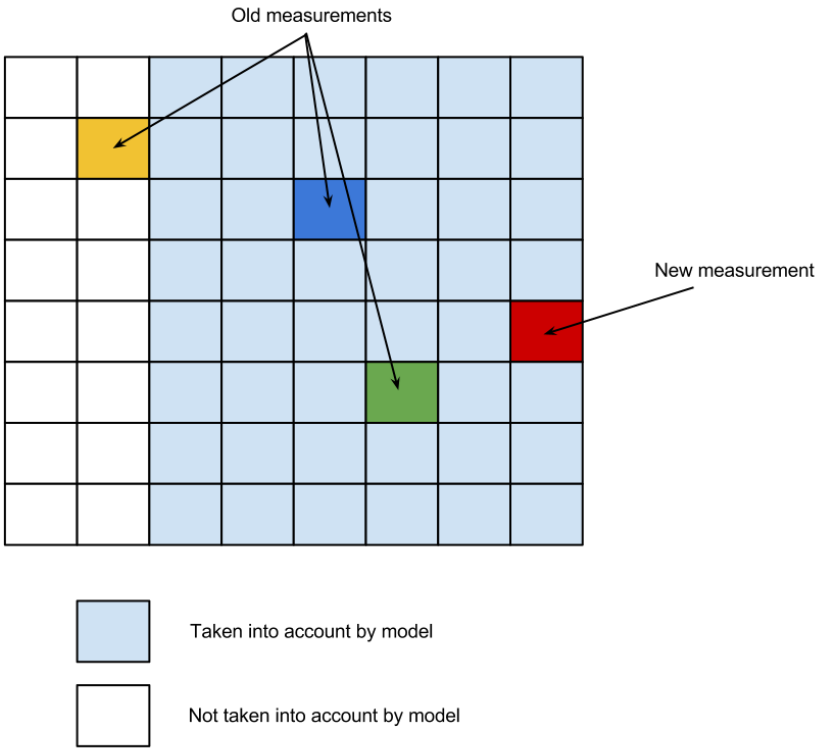

Figure 3.4.: The distance model - a new measurement arrives.

The interpolation of values in decibels however is not so simple as in figure Figure 3.7. Indeed, decibels being logarithmic units we have to transform every value using the following formula:

$$power.value = 10^{dB.value/20}$$

Only then can we find the *probabilistic values* by interpolation. We then have to transform these values into decibels again using the following formula:

$$dB.value = 20 * log_{10}(power.value)$$

The second model called "the sound attenuation model" will check every square in a radius of 50 meters of our measurement square (being careful not to go beyond the limits of the array). If there is already a measurement with a less valuable probability or no measurement at all it will build *probabilistic data*. We can change the radius of dissipation easily to change the behaviour of the model as needed. The new data will be built using the formula for sound attenuation over distance, which is :

$$value = source.value \text{ - } 20 * log_{10}(distance.from.source)$$

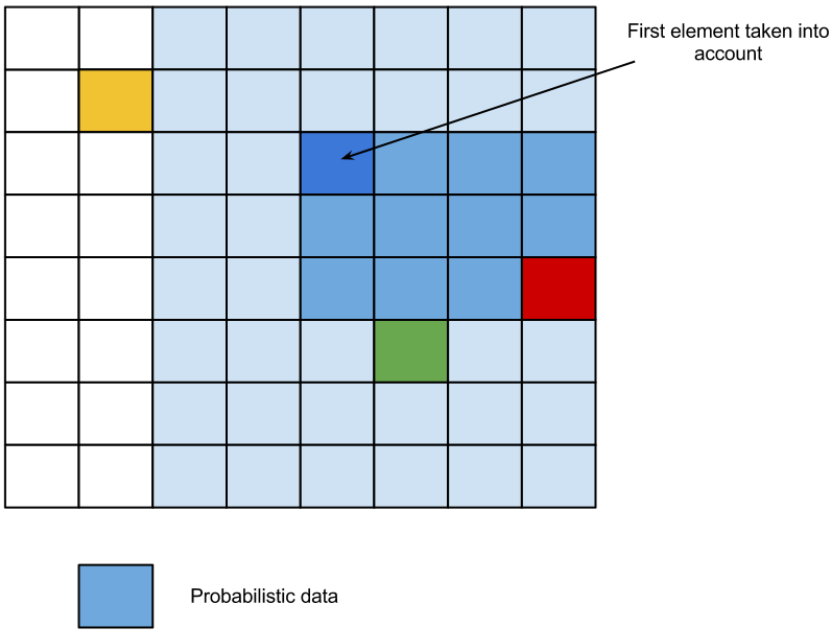

Figure 3.5.: The distance model - furthest previous measurement.

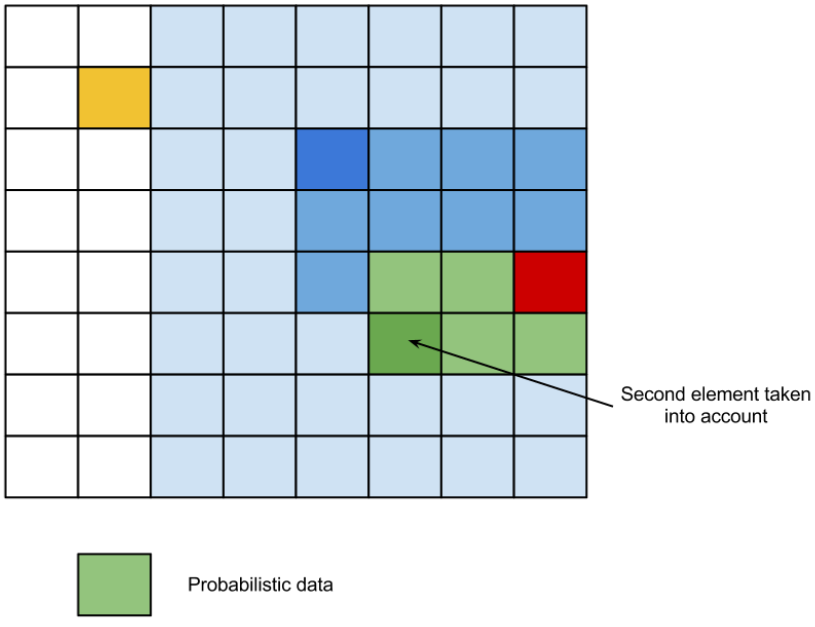

Figure 3.6.: The distance model - closest previous measurement.

The values are in decibels. We decided to apply the model only 50 meters around the measurement received by the web service because the accuracy of this model is pretty

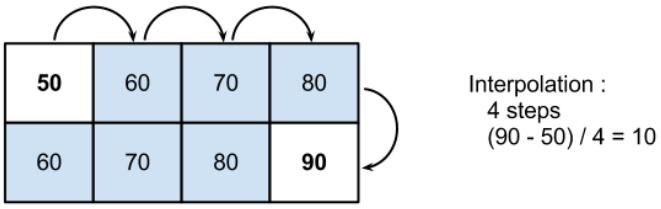

Figure 3.7.: A simple interpolation example of the distance model.

bad. We cannot be sure that there are no obstacles between our measurement and the probabilistic measurement we just built and the formula we use is very simple and does not take into account any kind of perturbation.
The model itself is bad, because we do not actually know where the source of the noise we just measured is, so the data should only be attenuated in some part of the surrounding area of the square and amplified in the other. This is however very difficult (impossible?) to implement in the given time of this project and would be the subject for a new master thesis. The probability value of this model is thus very low. Still we can always use the model just to check how it behaves in reality and see how good it is.

Figure 3.8 shows how the model works.

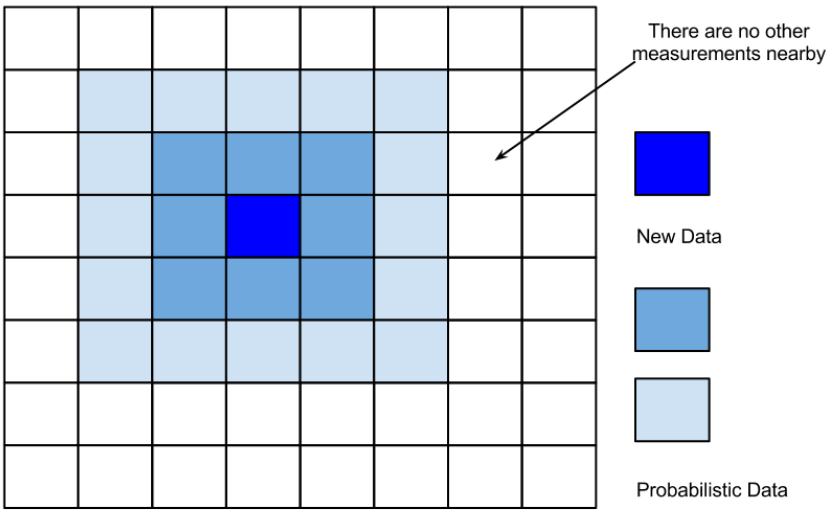

Figure 3.8.: The sound attenuation model.

The combination of these two models allows us to build a good amount of *probabilistic data* that we can use to populate the database and for the visualization part.

Inserting data in the database

Once every 30 minutes, we want to parse the temporary file, retrieve all values within and transform them into sets of *RDF triples*. We then want to insert these triples in the database and empty the temporary array.

For this task, we have implemented a PHP script that will be called using crontab. Crontab is a UNIX command that creates a table or list of commands, each of which is to be executed by the operating system at a specified time.
Our script will parse the temporary file. Each time it reads a measurements, it will extract the different parameters stored, calculate the latitude and longitude according to the position we are in the array, get the current date and time and create the set of RDF triples that are going to be inserted in the database.
Each measurement must have a unique identifier. Therefore, we have created a file containing a number corresponding to the next usable identifier. Each time an identifier is assigned, the value in the file is incremented by one.

The major problem in this part of the project is that *dipLODocus* has not yet been implemented with functions that would allow us to insert new data or to update old data. Such features will be added in the near future, but for now we had to find another way to insert our data.
Even without such functions, *dipLODocus* can build a new database from files containing *RDF triples*. Therefore, the temporary solution we will be using, will be to add our new sets of triples to such a file, and then to call the build function of *dipLODocus* just after (still using crontab) so that we have a new version of the database, containing the latest *RDF triples*.
This will have to be replaced as soon as *dipLODocus* will give us the possibility to insert data on the fly though.

Once the data is inserted, we want to call our time models. These models should be called at this moment for two reasons. Unlike the spatial models, they need data that has been inserted previously and not only data from the temporary array. It is also best to use the data from the temporary array we want to insert when we are sure no new measurements will modify it no more.

Time models

The first model is called the "last hours model". Once we have inserted our new measurement in the RDF triple file, we check if the database contains any measurement for the same square but 30 minutes before. We do not take *probabilistic data* into account. If it does this means that no interpolation is needed and the model is done. If not, we check for any measurement one hour before. If we do, then we will create a new set of triples by interpolating the values of the measurement from last hour with the ones of the actual measurement. If not, we check for any measurement one and half an hour before. If we do have such a measurement, we create two sets of triples by interpolation this time; one for one hour before and the other for half an hour before.

We could search for further measurements even farther back in time, but the more we go back and the more probabilistic inaccurate data we will create. Indeed, one could imagine that if the noise level for a given place now is the same as it was one hour before, then it was also the same 30 minutes ago. But if it is the same as twelve hours ago, the chances that it stayed the same during the whole time are smaller. Interpolation is more accurate if we interpolate two measurements that are close to each other, in time or distance.

Figure 3.9 is a representation of this model.

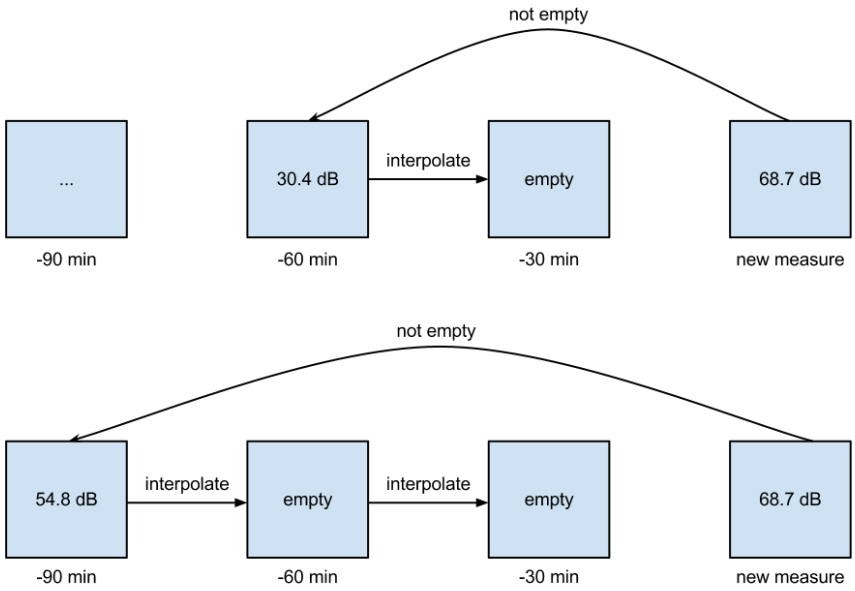

Figure 3.9.: The last hours model.

The second model is called the "last weeks model". This model searches in the database for any measurement for the same square two weeks ago. We still do not take *probabilistic data* into account. If all of its noise values (average noise and peak noise levels) are similar to the values of the current measurement, we can imagine that it was the same for last week. Therefore, we create a set of RDF triples for last week by interpolation.
By "more or less", we decided to compare the values with a maximum difference of 3 decibels allowed.

Figure 3.10 is a representation of this model.

The problem with both of these models is that we can create *probabilistic data* even if less valuable *probabilistic data* already exists, but we cannot delete this less valuable data. This is not a big problem, since we can retrieve both sets and take the more valuable into account, but still, we remain with unneeded data in the database. This will be solved once *dipLODocus* will be able to insert, update and delete triples.

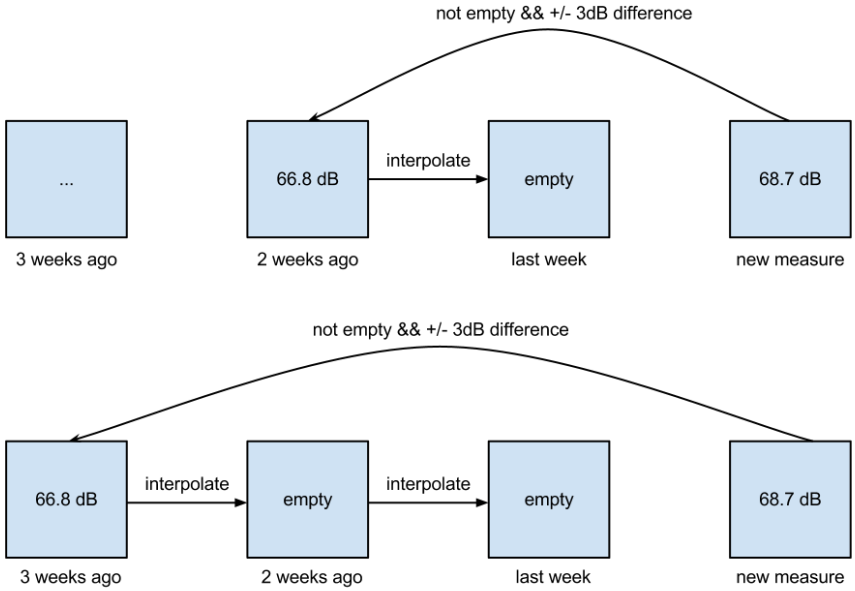

Figure 3.10.: The last weeks model.

Once the models have created and inserted new triples in the triple file, *dipLODocus* will build the database, thus finishing the 30 minutes cycle. Every new incoming data will be part of the next 30 minutes gap of time.
However building the database can become expensive in time if the file is big. Still, in the case of incoming data, it is the web service that will be needed, not the database. This could be a problem in the case where users can not access the database for the visualization part while it is building, however in our case the triple file not reach such important size and as soon as the data insertion is available on *dipLODocus* we will change the current behaviour.

Early model approach

The modelling part can seem easy in theory but it was quite hard to implement it.

Our first approach was to have a three dimensional array with an axis representing time while the two others would represent coordinates. Figure 3.11 and Figure 3.12 show this array. Such a representation was really hard to implement using PHP however and would have been bad in terms of ressource consumption. Indeed, PHP was not really meant to do this kind of work.

A good solution would be to use SciDB [10]. SciDB is a new open-source data management and analytics software system that is organized around a multi-dimensional array data model, a generalization of conventional relational databases that can provide orders of magnitude better performance for certain applications. SciDB is designed to store huge amounts of data distributed over a large number of machines.

However, it is still in development and we had troubles to configure and to use it. We were afraid that it would take too much time to integrate it to the system, and it seemed a bad solution to have two different database management systems (one for storing the temporary array and the other for the RDF data).

This modelling solution is not a bad one and should be taken into account in future work on the project and compared with the current implementation.

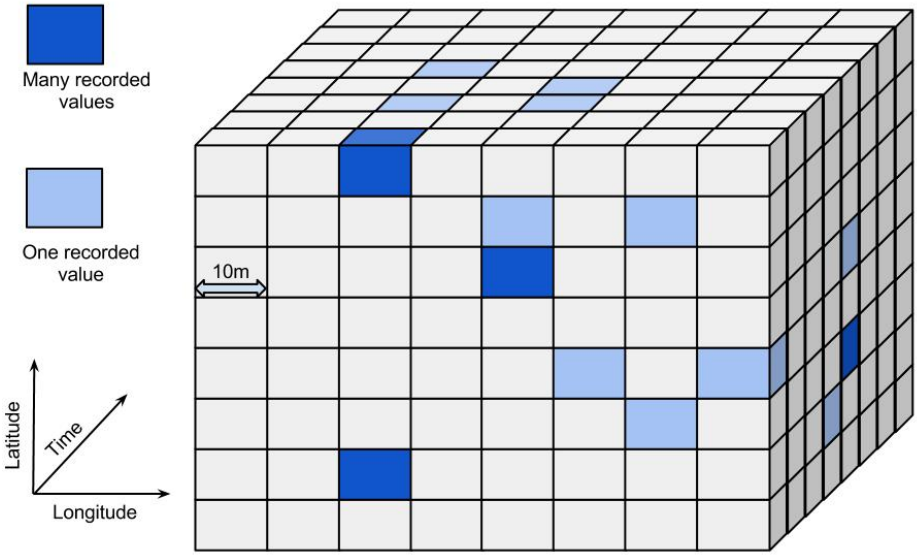

Figure 3.11.: Our first approach to models.

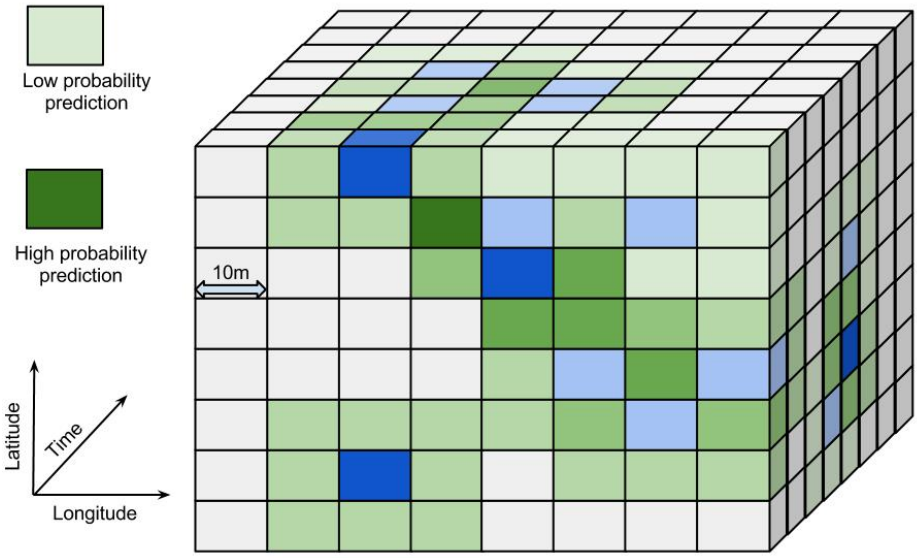

Figure 3.12.: Our first approach to models - interpolation.

3.3. The database management system

The database management system (DBMS) in our project is responsible for storing the data and making it available to the user through visualization.

in regard of the huge of amount of data we expect to have in our system, we need a DBMS that can handle such an amount and be fast to retrieve data. Indeed, if we want the visualization to be comfortable for the users the data they are looking for must be accessed in a reasonable time.

The storing process of the data can be long, we do not care that much.

3.3.1. RDF definition

The Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax formats. [8]

An RDF triple contains three components:

- the subject, which is an RDF URI reference or a blank node
- the predicate, which is an RDF URI reference
- the object, which is an RDF URI reference, a literal or a blank node

An RDF triple is conventionally written in the order subject, predicate, object. [9]

We chose this data model for our system because *dipLODocus* can handle it very fast.

3.3.2. dipLODocus

We chose to use *dipLODocus* [1] as our database management system.

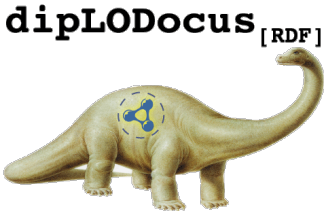

Figure 3.13.: dipLODocus.

dipLODocus is a new system for RDF data processing supporting both simple transactional queries and complex analytics efficiently. It is based on a novel hybrid storage

model considering RDF data both from a graph perspective (by storing RDF subgraphs or RDF molecules) and from a "vertical" analytics perspective (by storing compact lists of literal values for a given attribute).

It seems as a good choice for our system, since we are going to have to store an important amount of data that will have to be accessible fast for the visualization part. It is still in development though, which is good and bad at the same time.
The positive aspect is that it is easy to add new functions or modules but some of these modules or features we might need are not yet implemented and would take too much time for us to do in the time we have for this project.
The main problem we encountered is that *dipLODocus* as of now has no feature allowing to insert, update or delete RDF triples on the fly. We had to found other ways to insert our data as you can read in chapter 3.2.2.

The main functions we have added to *dipLODocus* during the project are an access via a socket that is waiting for commands and parameters, and functions allowing us to retrieve the values of our measurements for the visualization part. These functions will be called by the web service and will return it the awaited values.

The socket is waiting to receive a chain of caracters. Once it does, it compares it to a list of chains of caracters it stores. If it finds a correspondance it calls the corresponding function by passing the chain as a parameter.
We can create any function that receives the chain of characters as a parameter and extract useful variables out of it and code any behaviour we want it to have.

We have created many functions useful for the visualization. These functions usually receive coordinates as parameters and return corresponding measurement values.
We did not want our DBMS to act in any way other than to store data and retrieve data, therefore we do not make any changes to the data within the DBMS (such as calculating averages or making interpolations). The different functions are explained in detail in chapters 4.2.1 and 4.2.2.

3.3.3. Problems encountered - the insertion of data

In the beginning of the project, we had to use Virtuoso, another database management system able to handle RDF triples, but we soon discovered that we are limited by it since we do not have access to its source code and have to use the available features only. We noticed that a simple function like retrieving some values of a measurement needed a dozen queries to be performed. Therefore we prefer to use *dipLODocus*, which is faster and which we can shape as needed.

The main problem with *dipLODocus* however is that it is not fully implemented yet and lacks some useful features. We cannot insert data, delete data or update data on the fly. We can however build a database out of a file containing RDF triples. Therefore, the web service is responsible to add every new triple to such a file, and using crontab we can build the database anew every 30 minutes.

This of course is a temporary solution that will have to be changed as soon as the insertion features are implemented. Indeed, the bigger the file becomes, the longer the build phase will take to complete, thus possibly slowing the whole system.

4
The visualization

In this chapter we are going to describe the visualization part of our system. We are going to present the different solutions we have used and the options we have implemented. We will give reasons to the choices we made, show some results and propose some ideas for future improvement.

I want to give credit to my colleague *Jean-Gérard Pont*, who has worked with me on this part of the project.

4.1. What we want to visualize

Now that we have a database containing measurements we would like to be able to visualize the different kind of noise levels on a map. We would also like to be able to get some statistics out of the data we have.

A good idea is for our users to be able to access the visualization service through the web on a browser. Many interesting APIs already exist for different kinds of visualization so we could use a few of them to obtain good results.

We now have to ask ourselves what the user of such a system is really looking for and what possibilities we want to offer them. Possibilities are numerous and we cannot implement them all.

We obviously want to be able to associate coordinates with noise values. The only good way to represent coordinates is on a map. Numbers alone would mean nothing to a user. We can than represent the noise values by adding an opaque layer to the map or by

showing values in another part of the page.

The problem is that on a map we can easily display one measure per area but it would be difficult to represent any kind of statistics or many measures at the same time. To do so, we think an additional view would be useful. We chose to also have a graph view so that the user could see the evolution of the values over time, as well as a comparison of different values.

4.2. The elements of the visualization

4.2.1. The map

The Google Map API [4] allows us to integrate maps to our website and is free[1], so it seems as a good solution. The API offers an impressive set of functionalities[2].

We want to display the last recorded noise value for the area displayed on the map. To do so, we have to query our database by sending it the latitudes and longitudes at the borders of the area currently displayed. We have implemented a function in *dipLODocus* that will return us all the measurements delimited by these parameters. If we have many measurement for the same coordinates, the function returns the one having the best probability (that has been really measured or the one having been generated by the best model). In the case we still have many measurements, it returns the latest one.

Once we have all the measurements, which are composed of coordinates and noise values, we create a google.maps.Rectangle() object for each measurement, with sides representing 10 meters, the coordinates of the measurement as the center of the object and the value of the noise level as the level of opacity of the object. We want these rectangles (which are actually squares) to be transparent so we can still see the map underneath, so the opacity level will not go beyond a value of 50%.

Since we have average noise levels and peak levels, we need to add a set of two radio buttons, allowing the user to choose which kind of value he wants to be displayed. We chose to display peak noise values in red and average noise values in blue.
We have also added a scale on the left of the map so that the user can see to which value in decibels a level of opacity corresponds.

Each time, the user moves the map or zooms in or out the same process is called again. The latitudes and longitudes having been modified, we query the database again and recreate all the rectangles. But before we have to delete the previous ones, and therefore each of these objects has to be stored in an array.

We have however limited the updating of the map to a limit of zoom. Indeed, after making some tests, we discovered that though *dipLODocus* can retrieve a huge amount

[1] As long as we do not exceed a certain amount of map generation per day and do not charge our users.
[2] Zooming, plan/satellite/street views, possibility to add objects or layers.

of measurements very fast, the Google Maps API takes a lot of time to create and display them. Once we have more than a thousand elements, we notice a delay that makes the visualization uncomfortable for the user. It would be a good idea for future implementation to build bigger rectangles out of many measurements (according to the current zoom on the map), by averaging them. The process would still be fast but we could cover larger areas, but with smaller accuracy though.

The *dipLODocus* function could also be improved. Indeed, actually we parse all the measurements in the database to find the ones we need. This means we store all the measurements in the main memory. This could be avoided by adding a new type to our RDF schema. All the measurements of a given region would be of one type, and therefore, according to the parameters we would send to the function, we would only need to parse a certain type of data which would be a subtype of the main type.

```
1  <http://GetSound/SoundLevel> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
       www.w3.org/2002/07/owl#Class> .
2  <http://GetSound/SoundLevel_000_000> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
       http://www.w3.org/2002/07/owl#Class> .
3  <http://GetSound/SoundLevel_000_000> <http://www.w3.org/2000/01/rdf-schema#subClassOf> <
       http://GetSound/SoundLevel> .
4  <http://GetSound/SoundLevel_0> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http
       ://GetSound/SoundLevel_000_000> .
5  <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#year> "2012" .
6  <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#month> "8" .
7  <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#day> "06" .
8  <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#weekday> "3" .
9  <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#hour> "16" .
10 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#minute> "06" .
11 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#timestamp> "1346940360" .
12 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#latitude> "46.794083044316" .
13 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#longitude> "7.158331750784" .
14 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#probability> "8" .
15 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#averagePower0> "49.711" .
16 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#averagePower1> "49.711" .
17 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#peakPower0> "56.391" .
18 <http://GetSound/SoundLevel_0> <http://GetSound/rdf-schema#peakPower1> "56.391" .
```

Listing 4.1: The modified set of RDF triples - a new subtype of measurements to improve the query.

We have already started the implementation of this solution, but it was not yet finished at the time of writing this report. It should be the first thing taken into account in future work on the project.

4.2.2. The graph

Now that we have a map, we want to be able to see some statistics of a given area. To do so we decided to add a graph to our interface. It will display statistics according to the choices of the user concerning a given area.

Highcharts is a charting library written in JavaScript, offering an easy way of adding interactive charts to web sites or web applications. It currently supports line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange,

columnrange and polar chart types. [5] And just as the maps API from Google it is free for non-profit use. Since it is interactive, it offers possibilities to get more information by using the mouse over the chart and other things that makes it really an interesting solution.

We have added a set of three radio buttons on the interface, one for each three different representations we propose. These representations are:

- Day - We show the average of all the measurements of the area for every hour.
- Week - We show the average of all the measurements of the area for every day of the week.
- Year - We show the average of all the measurements of the area for every month of the year.

We could have added many other options, but this has to be discussed in future work on the project.

Both average and peak values are always displayed on the graph simultaneously and in the same colors as for the map; red for peak values and blue for average values. The axis of the chart change according to the option chosen and the values returned by the database management system.

The selection of the area is done by the user by clicking on any point in the map. The coordinates of the point are displayed on the left side of the chart and sent to the database management system. We have implemented a function on *dipLODocus* that will find the square containing these coordinates in the database and return the different noise values. The latest average noise value will also be displayed in a textbox above the coordinates of the point we clicked.

We could improve the selection of the area by the user by allowing him to select an area by dragging the mouse over the map or by selecting two points. This will have to be analyzed in future work because it might improve the visualization, but since we already use the dragging of the mouse to move the map it might possibly make things worse.

4.3. View of the interface

Figure 4.1 is a representation of our visualization interface.

A: The scale indicating the noise level in dB in regard of the opacity.

B: The overall average value of the clicked area.

C: The coordinates of the clicked area.

D: The map. The Blue squares represent the last average noise value measured for the area represented by the square.

E: The graph.

F: The map display choice between average or peak values.

G: The options for the graph display.

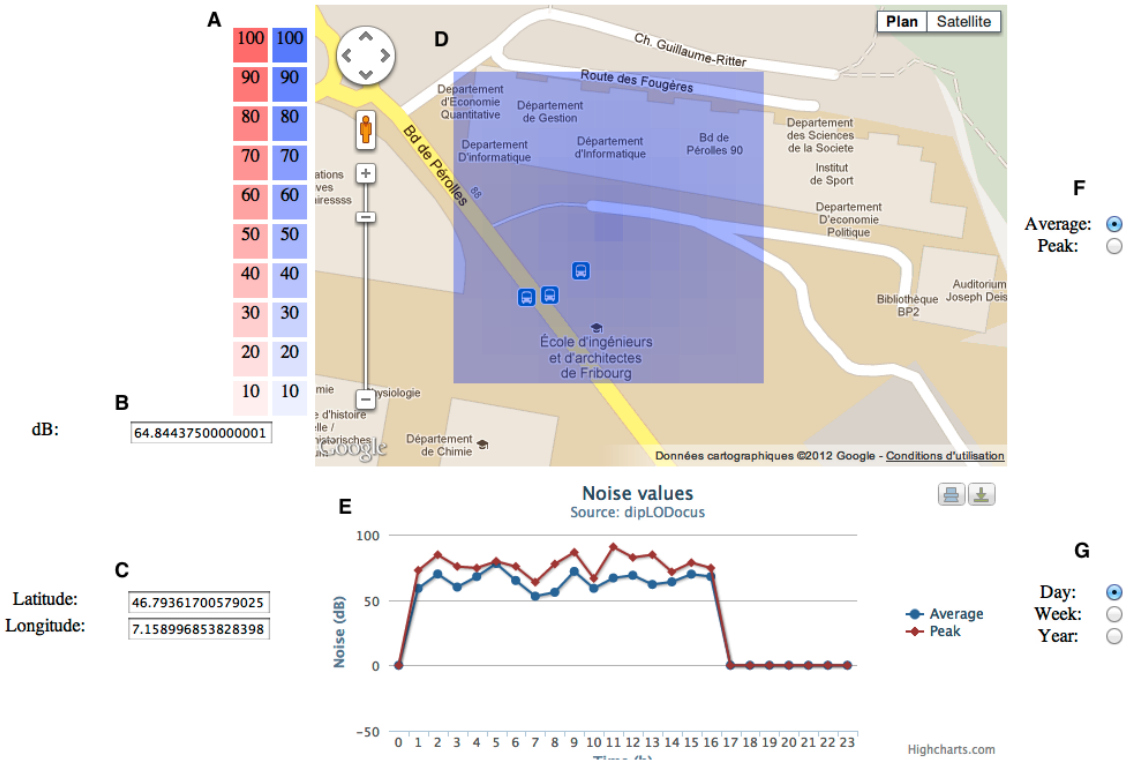

Figure 4.1.: Our visualization interface.

We are pretty satisfied by the visualization, since we can finally get something out of the system and have the feeling that everything that has been done before is useful and working well. We now have a fully functional system!

5
Testing the system

So now that we have a functional system allowing us to take measures, get and create data out of them, storing them in a database and visualize the stored data, we can run some tests and check if the results are satisfying.

5.1. What are we going to test?

We have to be sure that our system is accurate and useful for our users. We do not want to give them false information so we must check some parameters.

- First we want to check if the area where we make the tests will be the the area that will be displayed on the map.
- We also want to verify if the values measured by our IPhone application are representative of reality. This can be done by using a real sound level meter.
- We want to check if the data generated by our space models is correct.
- We want to check if the data generated by our time models is correct.

5.2. Description of the experiment

We have done the experiment on a saturday afternoon on a large parking (80x60 meters) of an industrial zone near Ecublens, VD. We have chosen this place because it is unoccupied during the weekend and the noise level is low and constant. Another advantage is that the lines drawn on the ground give us a good representation of distance. Indeed, a parking place is 5 meters long and 2.5 meters large and we need to take measures every 10 meters. Figure 5.1 shows the location of the place.

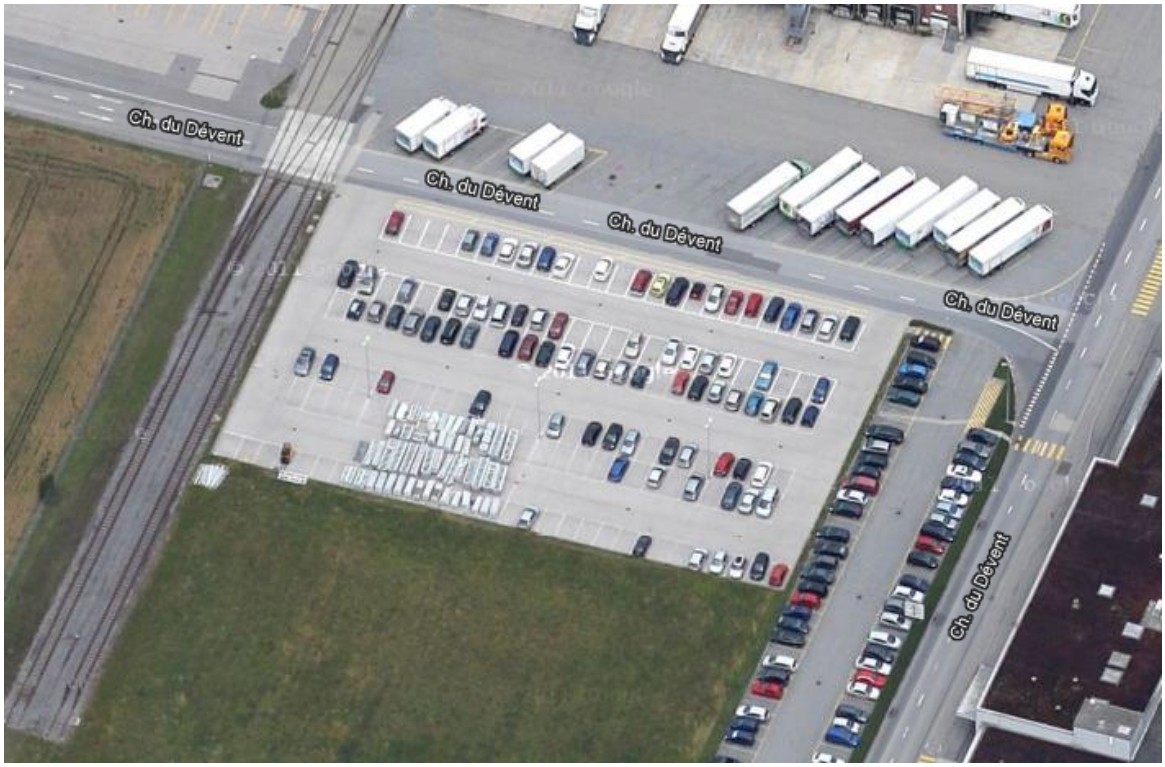

Figure 5.1.: The place where the data gathering for the experiment was made.

It would have been much better to test our system in an anechoic chamber without any perturbations, but we had to do it outside because of the GPS precision aspect and because we have to take measures of a large area.

The experiment was done as follows. The parking area has been divided into squares of 10x10 meters. We put 3 source sounds playing the same constant sound at different places. Then we took a measure for each square using the sound level meter. This was hard to do because values it displayed could change very quickly and the measures we have done are pretty inaccurate. Still by testing both our application and the sound level meter at the same time in a closed room with no disturbance the values were quite similar (the maximal difference was of 2.7 dB). The next step was to do the same with the IPhone. For this part of the experiment, we have created another IPhone application recording the sound and measuring it for 10 seconds, but instead of sending the values it would print it in a text box with the coordinates. After having measured the whole area,

we would start again. In the end we measured the whole area with the application 3 times.

We now have real data for the whole area that we can use. By inserting some of it in our system we can check if the *probabilistic data* that our *models* will generate is similar to the measures we have taken.

Figure 5.2 and Figure 5.3 show the values we measured with the positions of the sound sources highlighted. You can see to which position the square numbers correspond to in Figure 5.4.

Now, we verify that all the measures taken are consistent and can be used for testing our models. Figure 5.5 and Figure 5.6 show us that our measurements are consistent. We must keep in mind that while the average values are stable the peak values can vary greatly since only the maximal value reached is taken into account per measurement. Since we have done this experiment outside we should keep this fact in mind.

5.3. Results for the sound attenuation model

In order to test the model we take one of the measurements we have and give it to the system which will generate the *probabilistic data* we want to verify. It is unnecessary to do this operation on many different measurements because it will always behave in the same way and the sound does not vary differently in the same conditions.

Figure 5.7, Figure 5.8 and Figure 5.9 show the results we obtained using our system. We can see that the model would be good for average values if it knew the position of the noise sources. The values highlighted in blue are the value that are close to the real values (difference of 4dB max) while the ones in red have a too great difference with our real data.

5.3.1. Conclusions

The model doesn't work well with peak values. We can clearly see the values generated do not correspond to the real values. This is due to the fact that sound can vary alot in a short time and in an unpredictable way. The peak value is the maximal value measured for an amount of time and is not an averaged value and therefore can vary greatly.

The model works better with the average values and in a close range to the measurement used by the model. We can see that if the model knew where the sound came from it could produce good results all around the measurement if coupled with a sound amplification model. This could be a n idea for future implementation.

54.42	53.09	55.91	53.64	56.43	63.07	61.99	58.77
54.99	58.05	55.27	55.85	58.04	62.34	63.84	58.34
52.51	55.78	70.83	65.28	63.52	60.4	63.01	61.65
53.39	52.26	58.12	63.3	67.38	62.35	61.59	57.58
52.33	55.08	55.77	61.05	63.15	59.31	57.6	62.44
52.26	53.46	55.28	59.54	57.97	56.73	51.6	58.89

57.19	57.18	61.27	57.44	54.6	58.88	56.56	59.72
55.8	58.33	59.27	57.65	61.59	60.26	61.59	62.32
55.76	56.43	58.11	68.87	62.23	61.32	62.7	59.41
53.68	54.98	55.69	63.27	61.48	60.29	59.96	58.2
53.33	54.65	55.09	57.24	55	56.63	57.91	56.34
54.12	54.61	54.72	56.49	55.76	56.63	55.1	53.71

55.02	53.27	54.66	54.86	53.72	58.14	58.58	60.14
57.63	54.68	59.58	56	58.87	58.88	59.27	58.98
60.46	53.3	61.81	69.44	64.37	59.6	61.29	57.13
54.11	52.98	57.15	63.26	63.15	58.73	59.27	57.05
51.89	53.68	56.41	60.56	59.47	55.14	58.98	56.71
52.49	53.54	57.35	58.3	57.26	58.16	56.85	55.18

Figure 5.2.: The average values we measured.

60.59	59.44	70.04	60.04	69.46	71.54	67.13	65.78
61.28	65.83	61.35	68.58	62.89	68.85	71.06	72.47
59.93	64.31	72.85	68	66.89	66.75	68.59	65.28
60.33	59.32	62.97	67.54	69.67	73.46	69.2	78.79
59.98	69.97	61.18	65.17	66.9	64.17	68.22	75.49
60.64	68.56	63.14	65.84	63.82	61.58	58.89	71.46
64.58	68.54	71.69	65.7	62.65	65.85	64.79	66.22
62.2	70.76	66.54	63.98	67.17	69.13	68.9	69.17
63.68	71.23	64.33	77.7	68.64	67.03	70.12	67.52
69.24	68.51	68.45	67.08	72.75	71.07	65.43	64.2
72.45	67.25	65.06	75.64	61.58	62.79	64.41	62.22
70.69	68.34	66.13	68.09	62.45	63.58	61.34	61.1
72.21	69.1	63.04	61.59	62.46	64.44	66.14	68.38
69.21	60.33	63.83	63.25	67.19	66.64	66.11	66.57
67.11	59.9	67.25	71.12	75.62	64.21	70.7	65.1
63.45	64.3	62.2	67.13	70.72	64.09	66.65	67.68
59.72	59.48	62.05	64.2	65.84	62.64	65.4	67.53
63.96	62.15	63.41	69.14	65.07	64.92	64.69	70.35

Figure 5.3.: The peak values we measured.

1	7	13	19	25	31	37	43
2	8	14	20	26	32	38	44
3	9	15	21	27	33	39	45
4	10	16	22	28	34	40	46
5	11	17	23	29	35	41	47
6	12	18	24	30	36	42	48

Figure 5.4.: The positions corresponding to a square number.

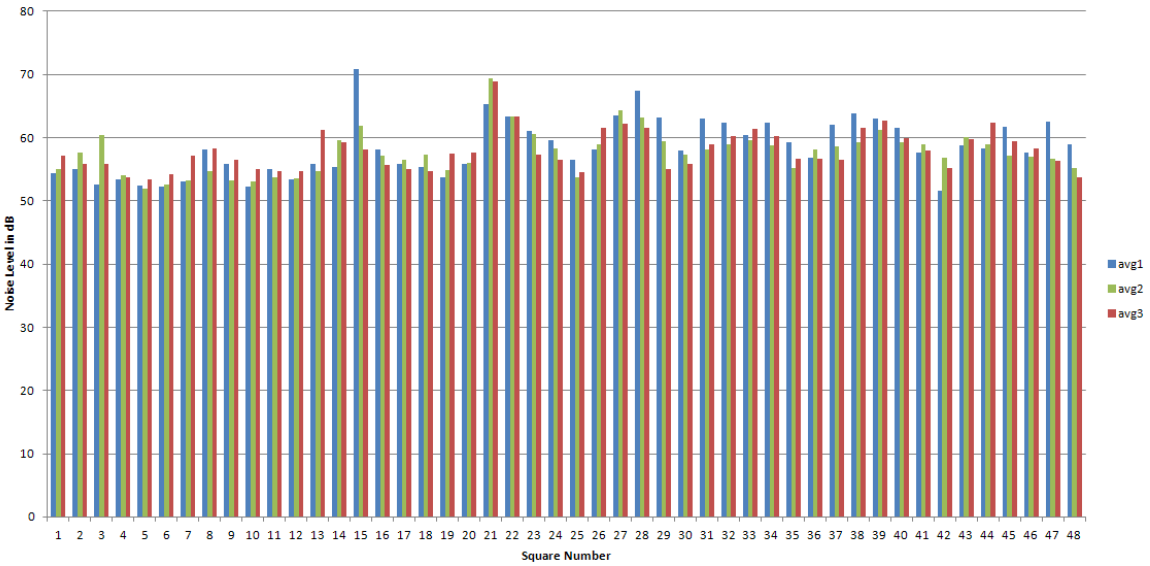

Figure 5.5.: The average values we measured.

5.4. Results for the distance model

In order to test the model we take three of the measurements we have and give it to the system which will generate the *probabilistic data* we want to verify. It is unnecessary to do this operation on many different measurements because it will always behave in the same way and the sound does not vary differently in the same conditions.

Figure 5.10, Figure 5.11 and Figure 5.12 show the results we obtained using our system. We can see that the values generated by the model are 63% under 4dB difference and 85% are under a difference of 6 dB for both average and peak values. The values highlighted in blue are the value that are close to the real values (difference of 4dB max) while the ones in red have a too great difference with our real data.

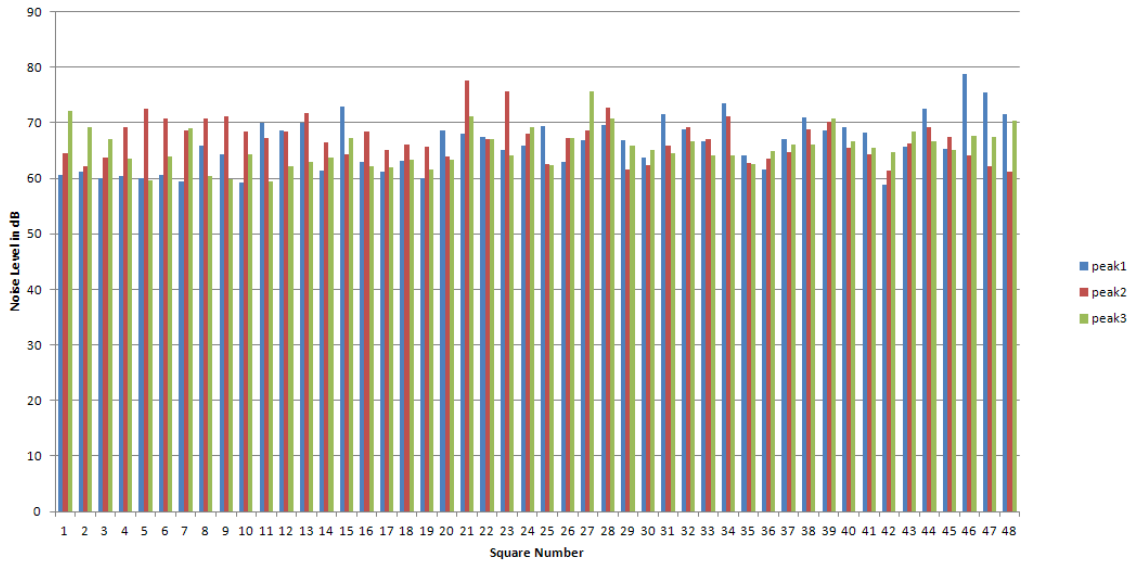

Figure 5.6.: The peak values we measured.

avg							
52.01	53.62	55.08	55.74	55.08	53.62	52.01	50.52
52.89	55.08	57.62	59.26	57.62	55.08	52.89	51.09
53.24	55.74	59.26	65.28	59.26	55.74	53.24	51.3
52.89	55.08	57.62	59.26	57.62	55.08	52.89	51.09
52.01	53.62	55.08	55.74	55.08	53.62	52.01	50.52
50.89	52.01	52.89	53.24	52.89	52.01	50.89	49.72

peak							
54.73	56.34	57.8	58.46	57.8	56.34	54.73	53.24
55.61	57.8	60.34	61.98	60.34	57.8	55.61	53.81
55.96	58.46	61.98	68	61.98	58.46	55.96	54.02
55.61	57.8	60.34	61.98	60.34	57.8	55.61	53.81
54.73	56.34	57.8	58.46	57.8	56.34	54.73	53.24
53.61	54.73	55.61	55.96	55.61	54.73	53.61	52.44

Figure 5.7.: The values generated by the sound attenuation model.

5.4.1. Conclusions

The model has a 85% success rate if we need to generate values with less than 6dB of difference and 63% if we want an error of less than 4dB. Therefore, we can say the model can be useful but is not perfect. It would have to be replaced by a more performant one if we want the system to generate realistic values.

5.5. Results for the time models

The time models are doing a simple interpolation when they find a missing value between two measurements. We have tested that the interpolation is done correctly during the development phase. Still if we want to see if the models behave in a realistic way, we need to take more measures over an extended period of time. This is something we sould do in future work on the project, but up to this point we did not have the time to do this.

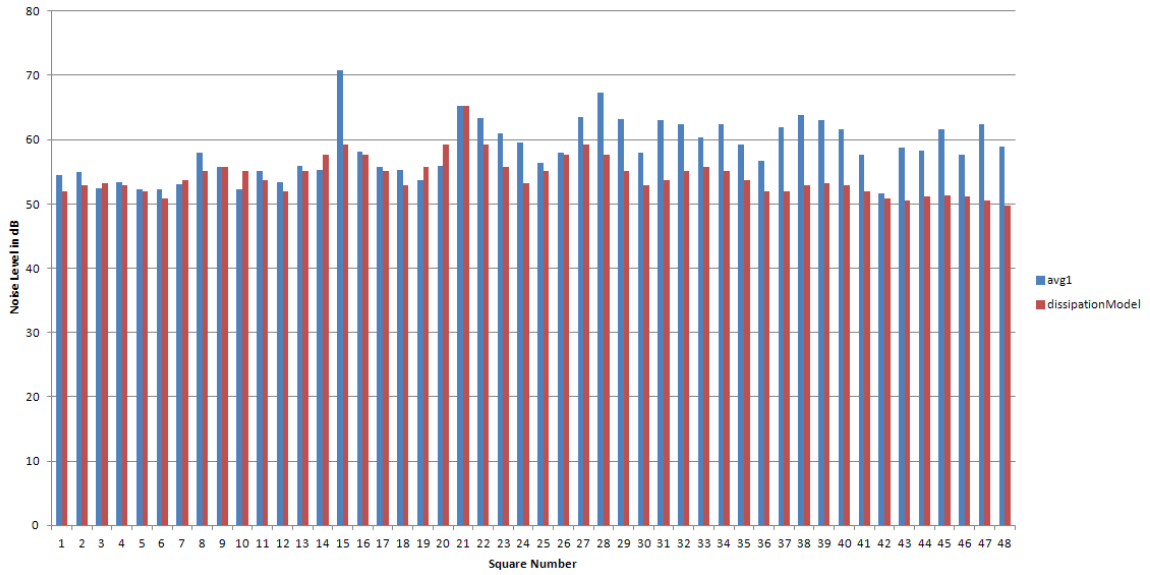

Figure 5.8.: The average values generated by the sound attenuation model compared to the real values.

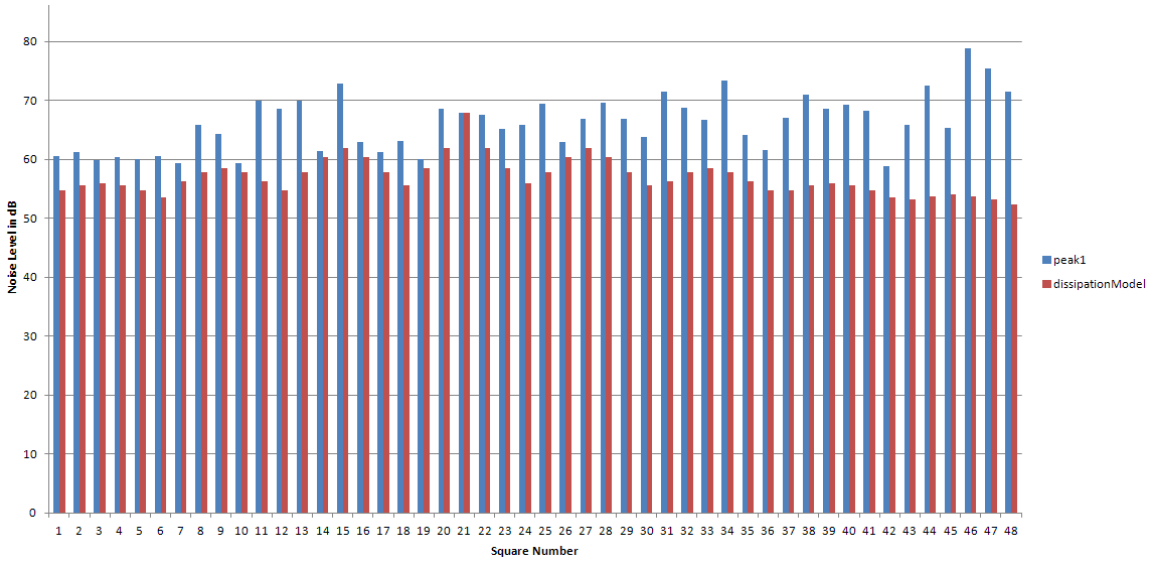

Figure 5.9.: The peak values generated by the sound attenuation model compared to the real values.

5.6. Final result of the Visualization using models

We can see in Figure 5.13 that our area is fully covered. The GPS precision of the IPhone is thus satisfying. We can see however that the visualization could be improved so that we could see the differences in noise levels better.

	57.75	59.36	60.17	61.89	62.92	63.84	
	55.78	56.67	57.48	60.17	61.89	62.92	
	56.67	57.48	58.22	61.65	62.25	62.81	
	57.48	58.22	58.9	61	61.65	62.25	
	58.22	58.9	59.54	60.3	61	61.65	

	65.86	67.18	68.32	69.34	70.24	71.06	
	64.31	65.86	67.18	68.32	69.34	70.24	
	64.64	64.95	65.26	68.47	69.19	69.86	
	64.95	65.26	65.55	67.68	68.47	69.19	
	65.26	65.55	65.84	66.81	67.68	68.47	

Figure 5.10.: The values generated by the distance model.

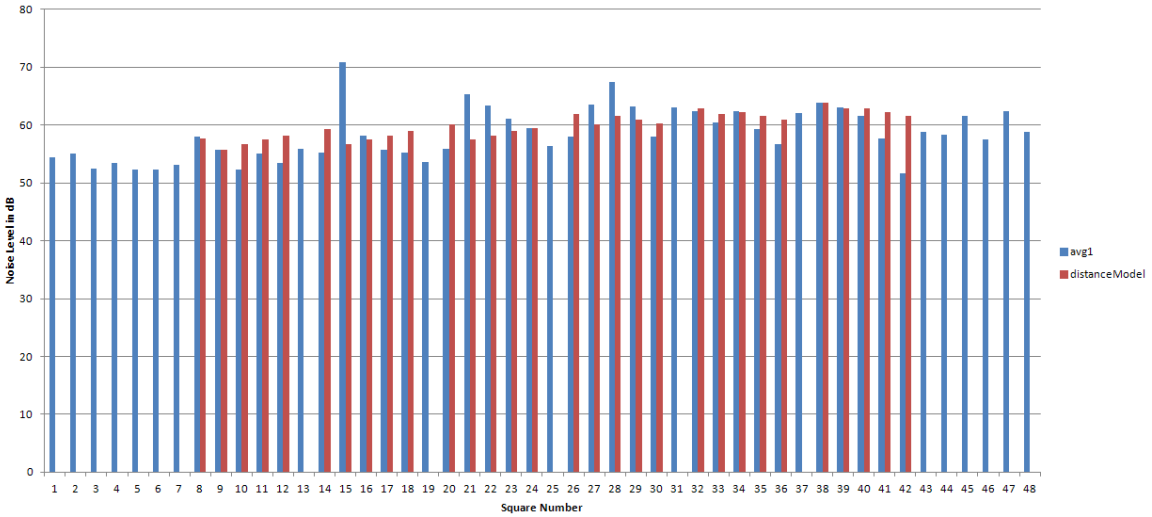

Figure 5.11.: The average values generated by the distance model compared to the real values.

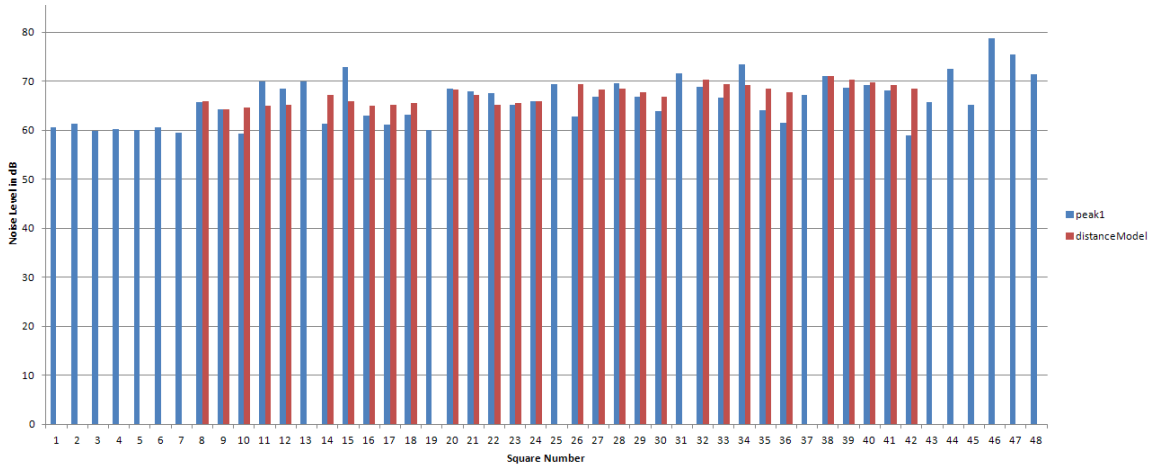

Figure 5.12.: The peak values generated by the distance model compared to the real values.

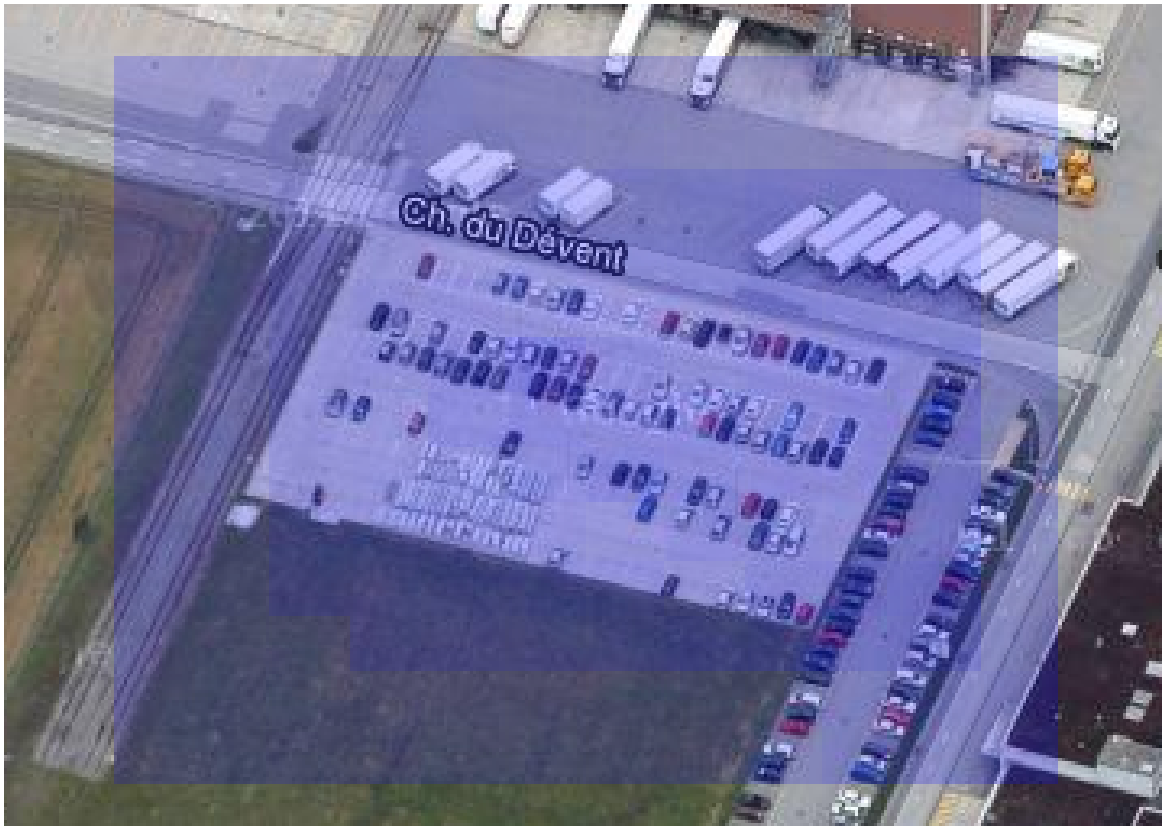

Figure 5.13.: The visualization of the area where we gathered the data.

6
Future Improvements

In this chapter we will describe the improvements we could bring to our system in the future.

6.1. IPhone application

We have a nice application taking measures and sending them to our system. But it would be good to add these features in a new application that would be interesting for the users. Indeed, no one will want to use the application in its current form. We decided during the project not to focus on how and when the user will use the application, this should however be considered in the future.

An idea would be to integrate the visualization to the application, so when the user uses it, it also makes measurements.

6.2. The web service

The web service could be improved in terms of concurrent accesses and security. We did not take these parameters into account during the project, but for the system to be functional and able to be used by many people we will have to work on this in the future.

The way the web service uses the database management system to store the data will have to be modified once *dipLODocus* will have insertion and deletion features implemented.

New models will also have to be added. The current models are simple and were made to show how the system could build *probabilistic data*, but we could now focus on complex models that would generate *probabilistic data* close in terms of value to realistic data. We have seen that just by finding a way to know the position of the source of sound could already improve the use of the models we have.

We have also started the modification of our RDF schema by adding subtypes for different areas of the region we cover. This way *dipLODocus* will not have to parse all the data it contains but only the data corresponding to a given type according to the areas we need to display. This will speed up the process and free alot of space in main memory (right now *dipLODocus* has to put the whole database in main memory, this will not be possible once the database grows really big).

6.3. The database management system

The system will work much better when we will have implemented the insertion and deletion features on *dipLODocus*. We invite you to read the article on *dipLODocus* [1] and to follow its development.

6.4. The visualization

The visualization is slowed by the amount of measures it has to display. Therefore, it would be a good idea to reduce the amount of measures by averaging the values of some areas when the zoom is low.

New options could also be added depending on what we want to offer to our users. We can visualize days, weeks and years right now, but maybe new options regarding week days or days of the weekend would be interesting.

The selection of the area we want to display on the chart could be modified so that the user can select more than one square. By dragging the mouse, he could select the his own boundaries.

The chart could also have a comparison option with previous displays.

6.5. Conclusions

There are still many interesting things we could do to improve our system. We should take care of the aspect of performance and security as soon as possible and leave the visual aspects for later.

7
Conclusion

Many visualization systems already exist for different kind of data. We wanted to create a system that could store huge amounts of data; some of it that would be generated by the system itself, and be faster than actual systems.

GetSound achieved these goals. With it we can populate a database using an application and models that will use the measured data to create additional data, thus filling the database with huge quantities of relevant data.
By coupling it with *dipLODocus*, we are able to store all this data and to retrieve it very fast.This system could be used to evaluate, demonstrate and make comparisons between *dipLODocus* and other database management systems.

We have discovered that by using models we can multiply the amount of data measured by our system's application several times. These models give average to good results and we are now certain that we can improve them and add new ones to obtain even better results.
Managing to achieve such a result could lead to a new way of gathering huge amounts of data and could be used not only for noise measures but for other kinds of data as well.

The visualization part of our system which is the interface that is used by users allows us to parse and visualize the data stored in the system. We have managed during the project to create functionalities for *dipLODocus* that will be used for returning the data needed by the visualization. This helped us getting an even better understanding of *dipLODocus* and to discover not only its good points but also the features it still lacks.

Finally, I am proud of this project and the results we achieved, and thank everyone who helped me during its development.

A
Common Acronyms

SPARQL SPARQL Protocol and RDF Query Language

RDF Resource Description Framework

URI Uniform Resource Identifier

URL Uniform Resource Identifier

W3C World Wide Web Consortium

PHP Hypertext Preprocessor

GPS Global Positioning System

B

License of the Documentation

References

[1] Philippe Cudre-Mauroux, Jean-Gérard Pont, Marcin Wylot, and Mariusz Wisniewski. diplodocus[rdf] | short and long-tail rdf analytics for massive webs of data. pages 778–793, 2011.

[2] Amol Deshpande and Samuel Madden. Mauvedb: supporting model-based user views in database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 73–84, New York, NY, USA, 2006. ACM.

[3] Free Documentation Licence (GNU FDL). `http://www.gnu.org/licenses/fdl.txt` (accessed June 30, 2005).

[4] Google Maps API. `https://developers.google.com/maps/` (accessed September 12, 2012).

[5] Highcharts JS. `http://www.highcharts.com` (accessed September 12, 2012).

[6] http://map.geo.admin.ch. `http://map.geo.admin.ch` (accessed September 12, 2012).

[7] Latitude and longitude geographical representation. `http://www.geographyalltheway.com/ks3_geography/maps_atlases/longitude_latitude.htm` (accessed September 11, 2012).

[8] RDF definition. `http://en.wikipedia.org/wiki/Resource_Description_Framework` (accessed September 12, 2012).

[9] RDF triple definition. `http://www.w3.org/TR/rdf-concepts/#section-triples` (accessed September 12, 2012).

[10] SciDB web page. `http://www.scidb.org/about/` (accessed September 12, 2012).