Carlos Manuel Oliveia Correia

Licenciado em Engenharia Informática

Improvements in the Geocoding Process in Organizational Environments

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientadores : Prof. Doutor Armanda Rodrigues, Prof. Auxiliar,
Universidade Nova de Lisboa
Miguel Marques, Senior Professional,
Novabase

Júri:

Presidente:	Doutor Nuno Preguiça (FCT/UNL)
Arguente:	Doutor Jorge Rocha (UMinho)
Vogal:	Doutora Armanda Rodrigues (FCT/UNL)

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

November, 2013

Improvements in the Geocoding Process in Organizational Environments

To all the parts of myself who keep fighting each other for dominance over my mind and make everybody else believe I am much crazier than I really am

Acknowledgements

I would like to express my deepest gratitudes to both my Advisers. To the Auxiliary Professor Armanda Rodrigues, her guidance and calm and joyful disposition were of great comfort during this thesis, and absolutely essential on every moment. And to the Senior Professional Miguel Marques, who guided me through and taught me about the realities of a business environment, all the while enduring my many... particularities. Without their help this surely would not have been possible.

In addition, I would like give my deepest gratitude to João Silva, and to the *Câmara Municipal da Amadora*. Their support was imperative for the realization of this thesis. The data provided was immensely important and extremely useful. Without their help this thesis would certainly lose some of its essence.

Additional thanks to both organizations which have supported my endeavours, the *Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa*, in particular, the Department of Informatics, and Novabase, the company which financed this project and which resources and environment have been greatly useful.

I would also like to thank my co-workers, who helped me through many hardships both related to this thesis and not. A special mention to Cosme Benito, both for giving me reasons to be angry at him and for letting me act on that anger, which was, in some ways, strangely therapeutic.

A huge thank you to my university friends, special mention to Paulo Ferreira, whose much too intricate thoughts always helped me to take my mind of things and to see them from other perspectives, and to Fernando Alexandre, who is also doing his thesis and has been a constant presence in my days, us both basking in the joys of finally overcoming a hurdle and cursing every living thing out of existence when Murphy's Law takes a stab at one of us.

A resounding thank you to my closest friends, Margarida Mendes and Ricardo Serra, who have been with me in both the harsher and more fond moments, and a thunderous applause to João Carvalho, who unexpectedly entered my life and has grown to be my one reliance on just about everything slightly related to me.

A single thank you to my family, to add to the gigantic pile of single thank yous I

have been giving them all my life, for their ever-lasting support.

And last but not least, a thank you to everybody else, because a world without the "everybody elses" would be no world at all, and even though they are rarely mentioned, they are always deserving.

Abstract

The current geocoding technologies are only able to handle addresses which fit a general case for the location in consideration. The more edge-case addresses are mostly ignored or wrongly geocoded, leading to imprecision and errors in the results obtained. To try overcoming this problem the current geocoding services accompany their results with confidence values, but the values and scales used vary between services, and are hard to understand by users without knowledge in the area, and, as we discovered, are not truly to be trusted.

Novabase aims to make available to organizations a geocoding service which allows the improvement of the quality of the results obtained by mainstream geocoding services such as Google and Bing. The objective is to give quality results in the cases where we can act and, not being able to do, falling back to the results of other geocoding services.

We pretend to handle addresses in areas where results are of inferior quality, either because the areas are not fully covered by the services or because those same services are not prepared to handle the address formats which do not match the general case (one example are addresses which are numbered by the use of *Lotes*).

The geocoding is executed in two steps. The first one matches the address with a knowledge base owned by organizations, in which we assume full trust of the quality. If the knowledge base returns a valid result, it is output with maximum confidence. When it fails, we fall back to using the mainstream geocoding services, and use their results for output.

Keywords: Geocoder, Geocoding, GIS, Address Analysis

Resumo

As tecnologias de geocodificação actuais apenas conseguem lidar com endereços que encaixem nos casos gerais. Os mais particulares são maioritariamente ignorados ou mal geocodificados, o que leva a imprecisões e erros nos resultados obtidos. Para tentar colmatar este problema os serviços de geocodificação existentes disponibilizam valores de confiança com os resultados calculados, mas os valores e escalas usados variam entre serviços, são difíceis de compreender por utilizadores sem conhecimentos específicos na área, e, como viemos a descobrir, não são verdadeiramente de confiança.

A Novabase pretende disponibilizar um serviço de geocodificação aos seus clientes que permita melhorar a qualidade dos resultados fornecidos pelos serviços de geocodificação de grande utilização como o Google ou o Bing. O objectivo é facultar resultados de qualidade nos casos em que tal é possível e, não o sendo, então usar os resultados de outros serviços de geocodificação.

Pretendemos então lidar com tipos de zonas cujos resultados sejam de qualidade inferior, por se encontrarem mal cobertas pelos serviços de geocodificação actuais, seja por estes não terem informação actualizada ou por os endereços dessas zonas usarem formatos específicos que diferem do caso geral (um exemplo é a utilização de lotes para definir os números dos edíficios).

A geocodificação é executada em dois passos. O primeiro pesquisa pelo endereço numa base de conhecimento em posse das organizações, na qual assumimos confiança máxima nos resultados. Se esta pesquisa completar com sucesso, o output será dado pelo serviço Novabase com o valor de confiança máximo. Se falhar,o endereço é passado para um dos serviços de grande utilização acima referidos, e os resultados deste serão o *output*.

Palavras-chave: Geocodificador, Geocodificação, SIG, Análise de Endereços

Contents

List of Figures

List of Tables

Listings

1

Introduction

1.1 Motivation

Both GIS (Geographic Information Systems) and geocoding are part of the everyday lives of most people. Setting a destination on a GPS (Global Positioning System), planning a trip on a computer, and looking up near pharmacies in a cell phone, are all examples of GIS and geocoding in action. Any time, anywhere and in such a simple way that most do not notice they are doing it.

As everything that becomes commonplace, it turns into an expected commodity from which users always expect better and more and, although in some areas the technology has been able to fulfill those expectations, in others it has been lacking. Geocoding services as they stand now are still unreliable; while they work well for general cases, there are many peculiarities which may lead to wrong results.

The aim of this project is not to replace current geocoding services, but to take advantage of what they can do and work in tandem with them to contribute to the improvement of tailored geocoding results.

1.2 Problem Description

Addresses are not an international standard. Each country has its own specificity in terms of addresses, each with its own edge cases and deviations from the norm. Current geocoding services will respond on international addresses, but will fail on edge case scenarios of specific countries. One example are newer quarter buildings in Lisbon, Portugal which have a new numbering schema.

Inaccuracies in the process prevent people and businesses from relying on current

geocoding services. High importance tasks will often require high levels of accuracy, which these services cannot yet provide under all circumstances. Distribution companies are a good example: the level of accuracy for many addresses in Lisbon is, at best, a region of delivery (e.g. a street) instead of a full address (i.e. a building). Another example is calculating the reach and efficiency of a distribution network, wrong locations affect those calculations and errors are propagated, generating false results.

To demonstrate, let us use the example of a fictitious company, Porcupine, a newcomer to the telecommunications industry. They have developed an application that allows their users to write down their address and check their coverage for Porcupine services. If the geocoding service cannot explicitly geocode addresses in (the also fictitious) Spike Street, the result of submitting addresses on this street to the application will be an approximation to the same location: the center of the city. Now let us assume that the center of the city is a location covered by the company, but Spike Street is not.

For the application users, this means that they are unintentionally tricked into believing they are covered, and adhere to the company's service only to receive a sub par experience. For the company, it is an unreported source of prejudice. All the users are reported as covered by the geocoding service, so the company has no need to improve reception in the area. In addition, when the company receives complaints from users, they will be unable to locate the problem and provide adequate solutions, leading to wasted resources and loss of trust from the consumers.

This kind of problem can be slightly eased by making use of confidence levels, which are labels or numbers that accompany each result and indicate the level of confidence the service has on the particular result it is providing. Even if the service is not able to pinpoint correctly a certain location, when the coordinates obtained come with a confidence level, organizations can act on that information and search to provide better services. In the example above, if confidence levels where present, customer service would have been able to better pinpoint the problem, enabling them to come with a plan of action while spending less resources.

Confidence levels served with geocoding results can help on mitigating the problem these inaccuracies carry. However, such analysis is difficult, even more so when comparing results of different geocoding services, since they often do not carry the same information. Current services are not rigorous when it comes to sharing confidence results with users. They are either too generic and uninformative, or delve too deeply into technical issues, which may prevent service users from understanding what they mean.

Geocoding results are not standardized among different geocoding providers, even if they carry the same information it may be presented in different formats.

When parsing a Bing geocoding response, the components of the provided address are indexed, and the application has direct access to them. The address comes in the following format:

```
1 [addressLine -> Main Street 17, locality -> Middle Town, postalCode -> 1234-567]
```

This means that if we want to know the postal code of the result, we only need to look into the `postalCode` property.

When parsing a Google geocoding response, however, we need to look at all the components to find the desired one. The address comes in the following format:

```
1 [(Main Street 17, street), (Middle Town, locality), (1234-567, postalCode)]
```

This means that if we need the postal code, we will need to go through all parts of the address until we find the one named `postalCode`.

This discrepancy on the formats used by the different services leads to a situation where switching services, or trying to use more than one, will often lead to conflicting results.

To understand why these types of problems occur in the first place, some knowledge in the workings of geocoding is necessary.

1.2.1 The Geocoding Process

Taken literally, geocoding means "to assign a geographic code" [GWK07]. However, for the purposes of this document, the terms "Geocoding" and "Geocoding Process", will be associated with the transformation of human readable addresses into geographic coordinates. This process, which is exemplified in Figure 1.1, varies in complexity, but can be simplified into two main steps: address standardization and translation of normalized addresses into geographic coordinates. In the normalization step, a series of operations are applied to the address, reducing it into a format appropriate for the next step in the geocoding process. The translation step, also called address matching, is where a series of algorithms are applied to the address in order to to acquire the coordinates for the address.

1.2.1.1 Normalization

The first step of the geocoding process. It is the transformation of a human readable address into a programmatic representation of itself. This involves applying some transformations [CCZ02] (such as lower casing the text, or removing punctuation) which convert the address into an internal format, rigidly defined, that the geocoder is expecting to find in the following step. Two main concerns are tackled in this step: name normalization and address components identification, as shown in Figure 1.2

Technically, it is possible to build a geocoding service without address normalization. However, at the very best, the resulting service would be very lacking in terms of results, since the absence of normalization implies that addresses are matched exactly as they come from the source, character for character. Most services will implement at least some degree of normalization. As the normalization applied to the addresses improves, so do the results obtained in the following step.

Name normalization is the substitution and sometimes correction of words with common synonyms with an equal equivalent. For example, in a street name the following

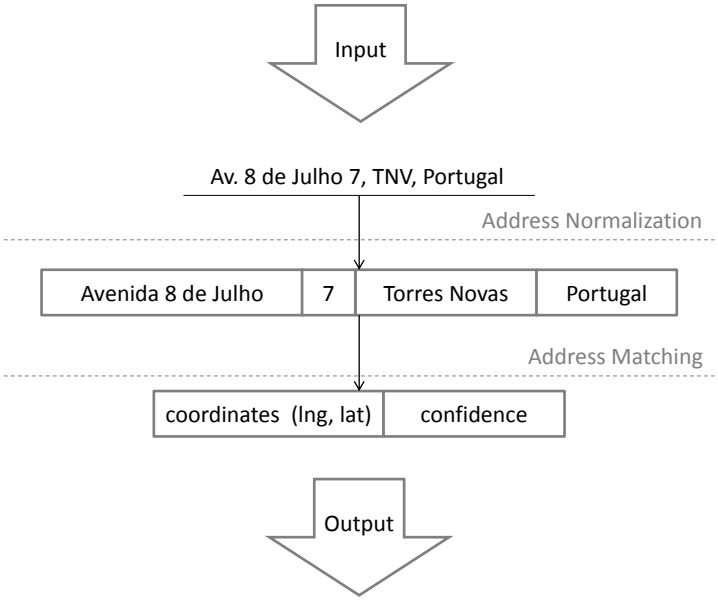

Figure 1.1: Geocoding example of a simple address

substitutions could happen: Avn. ➔ Avenue and St ➔ Street. The same is valid for the remaining parts of the address, such as locations and even street numbers: TNV ➔ Torres Novas, PT ➔ Portugal and nº7 ➔ 7.

This step can have various degrees of complexity [GWK07]. Some transformations could be considered easy, such as token parsing and search in lookup tables, which could potentially be enough for most kind of word substitutions, and others are much more complex, such as probabilistic methods [CCW04] and machine learning techniques [CC02], which could be used to correct erroneous input and even fill missing information.

The next step involves the identification of the components of an address. It starts with the result of the name replacement step and goes from that line of text, which represents an address, to the actual pieces that make it an address: door number, street name, postal code, and others. Each of these components are detected (if present) and stored for later use in the address matching step.

1.2.1.2 Matching

Translation of the normalized address into coordinates is performed using a variety of different algorithms. The most observed techniques being address lookup, hierarchical lookup and interpolation techniques, which will be described in the remainder of this section.

Address lookup is the simplest form of geocoding. A geocoder simply matches the normalized address with the information it contains in its knowledge base and returns the coordinates associated with it. If no match is found, other techniques are applied. One possible option is hierarchical lookup.

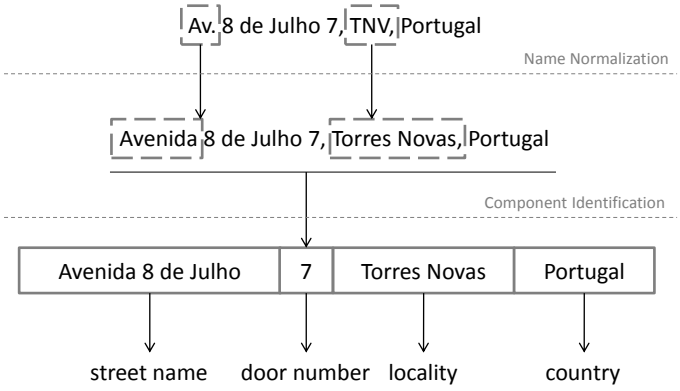

Figure 1.2: Example of address standardization

Hierarchical lookup relies on the fact that addresses follow a hierarchical structure. Take for example the address "Avenida Júlio Dinis 7, Sacavém, Loures, Portugal". There are four hierarchical levels present: street, two levels of administrative areas, and country. A geocoder that fails to recognize the street can try to recognize the next step in the hierarchy instead of failing, which in this example would be the first administrative area. As a consequence, however, the accuracy of the result is decreased, which is why interpolation techniques are used when there is enough information available. A concrete example of hierarchical lookup is the one used by Bing Geocoding API (Application Programming Interface), which includes reference to the use of this technique in the answers provided by the service [Bin13].

To provide better results, geocoding services may employ various techniques, such as linear interpolation, geometric interpolation, corner insets [Rat01, CT03] and street offset adjustments [Rat01]. Less common techniques include parceling and population distribution across areas [Gat89].

Linear interpolation is used when the geocoder can access information about street geometry and street numbers boundaries and positions. Let us take the street of the previous example, "Avenida Júlio Dinis 7", and assume we know the street geometry, that the street buildings are numbered from one to twenty and that odd numbers are on the west side of the street and even numbers on the east side. With this information we can approximate the position of the building numbered seven: it will be on the west side of the street since it is and odd number, and it will be seven twentieths into the street, since there are twenty houses. The result of the application of this technique can be seen in Figure 1.3.

An approximate position has been obtained, but since the information is based on the street axis, our coordinates point to the middle of the road, which is not correct. Street offset adjustments correct these cases. The street number is odd, which means that the building is located on west side of the street. Adjusting the resulting coordinates and moving them to the side improves the result of the geocoding procees by taking it off the

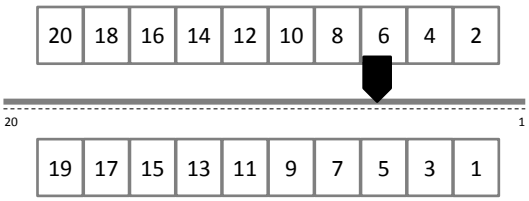

Figure 1.3: Geocoding with linear interpolation alone

road and into the locations of the houses. The new result can be seen in Figure 1.4.

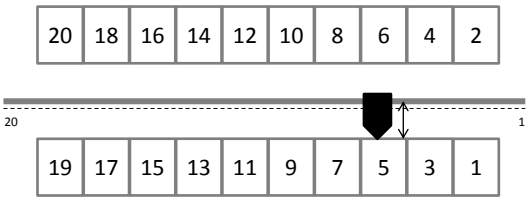

Figure 1.4: Geocoding with linear interpolation and street offset

This approach works well when buildings fill the whole street, but in streets starting (or ending) in a V-shaped bifurcation, houses often start later (or end earlier). In such cases there will be no buildings at the beginning (or end) of the street. To solve this problem, corner insets may be introduced. They force street numbering to begin (or end) only after a given distance and, in doing so, avoid the generation of results which locate buildings at street extremes. The final result is in Figure 1.5.

The previous algorithms have a pre-requisite of information. It is assumed that information on street axis and street numbering and positioning is present. However, it may happen that some part of this information is not available, in which case they cannot be used. When this happens, one solution, before rising in the hierarchy, is to use geometric interpolation, to obtain the centroid of the street [VMN01]. The centroid of a street is the calculated center of the street, each point of the street is averaged and the results is the centroid.

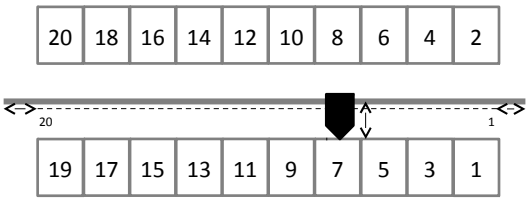

Figure 1.5: Geocoding with linear interpolation, street offset and corner insets

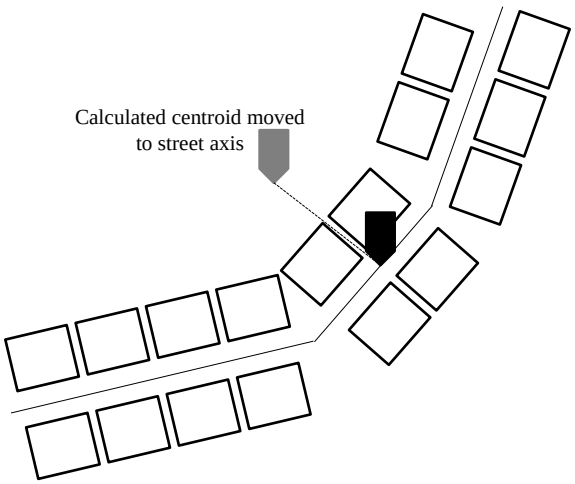

Figure 1.6: Geocoding with geometric interpolation

The centroid will normally be calculated far away from a point on the street, because streets are no rectilinear. Because of this a new calculation is performed to find the street segment which is closer to the centroid, and the middle of that segment is the chosen position. This position is then returned as the approximate result. An example of application of this technique is shown in Figure 1.6. Here, the calculated centroid is located almost behind the houses. To avoid this, the minimum distance between the centroid and the street axis was calculated, and the centroid was moved to the resulting position.

1.2.2 The problems

These and other techniques exist to improve the results of what is a very complex and error-prone process, and one of the problems of its complexity is that there are multiple points of failure.

Our research on the current geocoding services, presented in further Chapters, revealed some of those failures. The two most common errors where deduced to have their origins on out-of-date information and incorrect parsing of addresses.

1. **Specific areas where every address fails** are caused by out-of-date information or incorrect handling of address formats. This is normally seen in newly urbanized areas and is easy to detect, but hard to correct.

2. **Full streets are geocoded to the same location** when the service being used does not correctly parse the street address.

3. **A small number of addresses fail on unrelated locations**. This can happen when the services has incomplete information on certain geographic zones. It is hard to detect since only one or two addresses may be affected in the same area, making it hard to notice when examining locations in bulk.

The first problem is mostly a symptom of out-of-date information, it usually happens on locations which have been recently constructed (or reconstructed), and when the services have not yet obtained the new information. The time it takes for the new information to replace the old old varies between services and their data providers.

The second problem is a symptom of both out-of-date information and incorrect parsing of addresses. It may happen when new streets are build, and a new format address is applied to them, in which case the geocoding service will not identify the street. When instead of having a new street built, an old one is remodelled and the street numbering updated, it may happen that services which have the old information incorrectly geocode the addresses based on it.

The third one is rarer, and harder to detect. It normally affects only one or two addresses on unrelated locations, when, for some reason, incomplete information is obtained. One observed case was a building which had its numbering hidden from street view, and was not identified and included as a building in the street.

1.3 Context

Novabase is an international consulting and IT business which develops and integrates intellectual property with their partners and manufacturers. It offers business solutions in a wide array of areas, such as telecommunications and media, manufacturing and services, energy and utilities, aerospace and transportation, financial services, and government and healthcare.

Novabase clients own and use georeferenced data, but it is being underused. They do not possess the means to correctly analyse and extract valuable information from it, and this prohibits them from gaining insights which can be attained through spatial analysis.

There is a need to better explore the data these organizations possess. Insights on geographic analysis can greatly enhance the operations of the organizations, and are currently becoming a central focus of their departments, with more and more organizations requiring the geographic analysis capabilities.

1.4 Objectives

The aim of this project to contribute to the improvement of geocoding results of organizational entities. We provide them a customizable framework which can both leverage the organization's own intelligence and improve trust on the obtained results.

The target audience are small groups or single individuals within organizations who possess the potential to leverage geographic information but do not possess the required know how to accomplish it. The application will fill these users needs by opening the possibility to better understand their own information in its appropriate context, without the need to understand the GIS machinery behind it.

Our intention is to help improve the overall quality of the current geocoding services and technologies. We do not intend to replace the current set of solutions available, but to build an application which works with them to improve results. With that in mind we defined the following objectives:

- Allow the geocoding of a big number of addresses at the same time (batch/bulk geocoding)
- Decrease the barrier of entry and learning curve by providing abstractions over the GIS technologies
- Enabling and facilitating the use of information the organizations already possess
- Allow easy integration of new geocoding services into the process, thus allowing easy cooperation with future technologies
- Give access to the the results in such a way that facilitates its integration with other applications
- Permit quick and contextual visualization of results through the use of maps
- Allowing effortless detection and correction of geocoding problems
- Eventual inclusion of the resulting application both in other projects of Novabase and with their clients
- Provide great usability to facilitate the use of the application (This objective was posteriorly dropped due to time limitations)

1.5 Contribution

We hope to improve and contribute to the use of geographical information and spatial analysis within organizations. Transforming addresses, which organizations already have associated with their other information, into geographic coordinates is the first step which opens a myriad of possibilities.

Our solution is able to use as source various geocoding services already present, bringing the best of their results to a common application. One of its greatest advantages is its ability to leverage the information already owned by organizations and use it in cooperation with the results of the currently available geocoding services.

By bringing together the best results of the current services with owned information, the results can be improved upon without lost of quality, something which was previously hard to do, when the choice was between using either of them, instead of using both.

The feedback capabilities included in the solution enable a greater trust on the quality of the results. Before it was difficult to perceive the quality of the results, but with

our feedback system the application is able to detect problems before hand, and warn organizations about what it finds. This enables users to verify a big number of results with minimal effort, while still being able to pinpoint small irregularities which would otherwise be hard to detect. The biggest advantage of this feature is that it can be used with the results of mainstream geocoding services as well as the ones which comes from the knowledge base. This means that even without a knowledge base, quality and trust in the results of these services can still be improved upon.

By enabling organizations to use their already owned data and making them aware of the quality of the obtained results, we aim to improve their trust on GIS, thus propagating its wide-spread adoption in every organization.

2

State of the Art

To better understand the current standing of geocoding technologies Novabase required two different studies. One side objective of these studies was to allow us to grasp some of the fundamental concepts and to recognize common pitfalls.

One of the studies examines geocoding applications which are, at least partially, freely available for use by users. The applications are tested in various areas and results are compared between them. This allows us to considerate which are the practices that are worth emulating, and which we should avoid.

The second one studies two geocoding service APIs: Google Geocoding Service API and Bing Maps Geocoding Service. Both APIs results are analysed and evaluated, in an attempt to distinguish which would better server our purposes. Novabase showed interest in using the Google API, but if this analysis revealed the Bing API to be overall better suited, a move from one to the other would be done.

The conclusions derived from these studies are presented in the end of the Chapter, in Section 2.3, Discussion.

2.1 Application Analysis

The current market of geocoding services was studied with the objective of better understanding the current status of the technology. From this research we tried to learn which were bad practices we could try solve or avoid in our planned application.

Applications were selected from a wide variety of case studies, but for the deeper analyses described here, they add to fill three requirements:

1. **Ability to geocoded addresses in bulk** We are testing in an organization environment, for the use case where they need to geocode a big group of addresses
2. **Ability to collect or visualize the results** Because we need someone way to verify the quality of those results.
3. **Ability to geocode international addresses** Many geocoders initially considerated only allowed for US addresses, but sample tests were composed of Portuguese addresses

After the filtering we were left with five remaining applications:

- BulkGeocoder [Bul13]
- Find Latitude and Longitude [Fin13]
- Batch Geocoder [Bat13]
- GPS Visualizer [GPS13]
- Topo.ly [Top13]

The applications were tested on three aspects which were deemed to be the focus of our solution by Novabase: quality of results, features available, and usability.

Quality of results evaluates if the applications are able to geocode the given addresses or if they fail, giving no results. It additionally evaluates whether the successfully geocoded addresses are actually correct or not. There are many case in which geocoders may give a wrong result, due to reasons described in previous Chapters, and these should not be considered as correct or valid.

Availability of features tested for additional features besides the geocoding of addresses. We recognize that there is much value to be gained from whether or not applications do something else to the results, such as showing them on a map or presenting them with a confidence level.

Functional aspects are not the only ones which influence the evaluation that users make of applications. Usability is a big part of it, and, unless users are severely constrained, they will not choose an application with poor usability (even if it has slightly better results) over one with rich usability [RDA01].

Each area is discussed in detail in its own section: 2.1.1, Geocoding Results Analysis, 2.1.2, Feature Analysis and 2.1.3, Application Usability, respectively.

2.1.1 Geocoding Results Analysis

To test the quality of the results of the geocoding of addresses we produced two test cases. The first test case contained 1500 standard addresses, which is to say that no edge cases or newly constructed areas were included. The expectation was that the chosen

APPLICATION	STANDARD ADDRESSES		UNCOMMON ADDRESSES	
	SUCCESS RATE	HIT RATE	SUCCESS RATE	HIT RATE
Bulk Geocoder	98,80%	97,67%	100,00%	8,47%
Find Latitude and Longitude	97,80%	97,53%	100,00%	6,23%
Batch Geocoder	100%	97,93%	100,00%	8,49%
GPS Visualizer	99,67%	99,60%	100,00%	8,01%
Topo.ly	100,00%	99,27%	100,00%	10,74%

Table 2.1: Quality of the researched applications' results

applications would have no difficulties in geocoding these addresses, and that accuracy rate would be very high.

The second test case is more difficult. It contains 1117 addresses, all of which have some of the following properties:

- The address format is not the conventional for the country standards
- The area has been recently built or rebuilt, so information is likely to be out-of-date
- The area is cold, which is to say that is not very commonly searched for or otherwise interesting

The expectation for this file is that applications will have a hard time geocoding it, failing many results or otherwise appointing them wrong locations.

The results of the analysis are in Table 2.1.

The SUCCESS RATE column stand for the percentage of addresses which were geocoded. This does not mean that the addresses was mapped to its correct location, it indicates only that the geocoder gave some kind of result, even if wrong.

The HIT RATE column makes the distinction. It contains the percentage of addresses which were both successfully geocoded (the percentage of the previous column) and correctly located. This percentage is relative to the previous one. For example, if one geocoder has a 50% SUCCESS RATE and a 50% HIT RATE, this means that only 25% of the addresses were geocoded to their correct locations.

As predicted, the first test case, composed of standard addresses, showed fairly good results. The application with the worse results was Find Latitude and Longitude, which failed in geocoding 2,20% of addresses. In this metric, the first three applications, Bulk Geocoder, Find Latitude and Longitude, and Batch Geocoder performance approached 98% while the other two, GPS Visualizer and Topo.ly approached 100%.

The second test gave some initially unexpected results, with 100% success rate, and no addresses failing. Further investigation led to the conclusion that this is due to the fact that the second test, despite being based on more problematic addresses, included streets which are well known to the services, which caused the no fail success rate. The first test case contained a much greater variety of street names, some of them unknown to the services, which led them to fail.

Nonetheless, when examining the results were led back to our expectations. Only around 10% or less of the addresses are correctly geocoded. What happened with most applications was that they recognized the streets in the addresses, but failed to provide correct building locations, either due to having no information on the actual disposition of buildings in the area, or because the parsing of the street number failed.

In most cases, in addresses with complex door numbering, such as 2.06.1.A, the number was reduced to just the first or second number, and the remaining discarded. This led to all addresses with similar door numbers being geocoded to the same position.

For undocumented areas the most common error was decreasing the precision of the street name. What we here call "decreasing of the precision of the street name" can be better described by an example. Think of the the address `Street Channing and Crawford`, which the geocoder does not recognize. What was observed was that some geocoders would drop parts of the address if that would increase the HIT RATE, and instead give results in `Street Channing` or `Street Crawford`, if they existed. This lead to a higher SUCCESS RATE but a lower HIT RATE.

2.1.2 Feature Analysis

In this study, key features were identified and evaluated for all applications. These features were chosen based on the interests Novabase had for its solution. The results can be found in Table 2.2, and the explanation of the features and their reasoning for selection is in the following list.

- **Provider** The name of the geocoding service the application uses. Since we intend to include the use of mainstream geocoding services in our solution, it is interesting to see how applications which also use them fare in their results.

- **Input** The formats in which the application accepts the information. Acceptance of many formats of data is considered important because it removes work load from users. The target users are businessmen which do not care for the problems involved in this, and expect their information to be simply accepted. Difficulties in this area would be considered a big downside for applications.

- **Output** the formats in which the application returns the geocoded information. This follows the same rational of the previous point, the most important point being that the output is expected to be in the same format as the input, even though sometimes it is useful to have options which allow additional formats.

- **Confidence** Whether the application accompanies its results with confidence levels, and what kind of levels it uses. This is important to increase trust on the final results. It is widely known that geocoding results are not always accurate. The presence of confidence values may be the deciding factor on whether users trust the results of the application or not.

APPLICATION	PROVIDER	INPUT	OUTPUT	CONFIDENCE
Bulk Geocoder	—	CSV & Excel	Text	Textual
Find Latitude & Longitude	Google	Text	0-9 by region	
Batchgeo	—	Tabular Text	Map	—
GPS Visualizer	Various	Text	Various	—
Topo.ly	—	Tabular Text	—	—

Table 2.2: Features of the researched applications

Of all the applications, only two of them discriminate their sources. Find Latitude and Longitude uses the Google Geocoding API, and GPS Visualizer gives users the option to choose either Google or Yahoo! BOSS PlaceFinder. Some of the applications considered in the initial phase of the research also used the Bing Geocoding Service or the MapQuest Geocoding API. Unfortunately they were not appropriate for testing. Some lacked the necessary features (e.g. not allowing batch address processing), others were limited to certain geographic areas (e.g. North America).

Bulk Geocoder is the only application supporting direct file upload, with the limitation that it must be in the CSV format or, if users require it, an Excel file sent vial email to the application's team. After uploading the file, users have the ability to specify which columns of the file contain relevant information (e.g. which contains data on the address, locality or country). All other applications require text input through web forms. Find Latitude and Longitude accepts a syntax were each line represents a whole address, while Batchgeo and Topo.ly require a syntax similar to CSV, but where values are separated by a tab character instead of by a comma. GPS Visualizer and Topo.ly require the headers to have appropriate names, such as "Address", while Batchgeo follows the example of Bulk Geocoder, and allows users to specify which columns contain which information.

The output of the applications is more diversified. Bulk Geocoder provides results in the same file format as they were sent, either CSV or Excel. Find Latitude and Longitude and GPS Visualizer deliver simple text, both using CSV syntax. Batchgeo is the only one which does not allow direct access to the geocoded information, instead it returns a URL. This link is to their domain, and leads to a page where a Google Map is displayed with the geocoded addresses. A relevant feature of this page is that it enables the manipulation of the appearance of the markers used to visualize the adresses, according to the data initially provided. It is even possible present additional information, which was not relevant for the geocoding process itself. GPS Visualizer is the most flexible application in this metric, it allows output in a myriad of formats: text with CSV syntax, geographic file formats such as KML or GPX, and various image formats such as JPEG or PNG.

Confidence levels are only provided by two of the applications. Bulk Geocoder attributes textual confidence levels such as *Building*, which means that the location is associated with a particular building or house, or *Address*, meaning that the position refers to somewhere on the street of the building, but not on its exact location. The numeric levels returned by Find Latitude and Longitude can be associated with a value of confidence in

APPLICATION	INTERFACE	INPUT	GEOCODING	OUTPUT	TOTAL
Bulk Geocoder	Neutral	Good	Neutral	Good	Good (2)
Find Latitude & Longitude	Neutral	N/A	Neutral	Good	Good (1)
Batchgeo	Good	Good	Good	Good	Good (4)
GPS Visualizer	Bad	N/A	Good	Bad	Bad (-1)
Topo.ly	Good	Neutral	Bad	Good	Good (1)

Table 2.3: Usability scores of the researched applications

the result, from 0 to 9, where 0 means that the geocoding failed and 9 that the result has precision to the building level.

2.1.3 Application Usability

Four usability features were analysed for each application. These four were determined to be the ones which would have the bigger impact on the evaluation of users. They are key points and users are certain to focus their attention on these areas.

A summary of the results are presented in Table 2.3, where each *Good* adds one point to the classification of the application and each *Bad* subtracts one point. The description of the meaning of the features (table columns) is as follows:

- **Interface** Encompasses several aspects of the application such as look and feel, ease of use and navigation, and error reporting and correction;
- **Input** Deals with everything the user has to do until the application is able to start the geocoding process, such as file uploading and structuring;
- **Geocoding** Analyses user control of the geocoding process, for example, monitoring and network resilience;
- **Output** Examines data retrieval and visualization capabilities of the application.

The following sections analyse the applications according to the chosen features. The analysis takes into account that a more feature-filled application will be more complex to develop and handling, providing more space to fail, while a less featured one will be able to efficiently do what it does better.

2.1.3.1 Bulk Geocoder

The interface demonstrates both good and bad practices, earning a Neutral score. It enables quick access to the geocoding process, even if with a lightly cluttered interface which easily distracts users. The instructions are clear and easy to follow but indistinguishable from normal page text. There is no indication of which fields are required or not, and users are only notified when they try to start the geocoding process and fail due to incomplete requirements. Each field does, however, present information on what content it is supposed to be filled with.

The input step is one of the fields which earned a Good score. CSV files are the only ones supported for upload and the operation is well designed, with progress indicators and error handling. There is a step that allows the users to specify which columns contain which information, after which the geocoding process initiates.

The user does not, however, have any control over the geocoding process. There are no progress indicators or time estimations of any sort. Once the process starts, the only thing users can do is wait for an e-mail containing a link to the results. The only indication available is a note on the page, which refreshes every ten minutes, informing the user that the process is still ongoing. Despite that, the process is not affected by network instability, which gives it a ranking of Neutral.

Post-Geocoding improves the overall score, receiving a Good rating. Geocoding results come with a confidence level, showing the precision of the results in text form (Building, Address, etc.). The results are safe on the application's servers and can be verified before download. Unfortunately, there is no way for users to reach their results through the application, this is only possible by e-mail, when the geocoding process ends. Results can be downloaded in either CSV or Excel files after payment, which is based on the number of addresses processed. The downloaded files include all the original information, plus three additional columns: latitude, longitude and (optionally) confidence.

2.1.3.2 Find Latitude and Longitude

This Interface was rated as Neutral. It includes step by step, clear and easy to follow instructions. All the information about input of addresses and output of coordinates is described in detail and in an intuitive way. Although simple and concise, the interface is also disorganized and overly aggressive, which may scare potential users. Going through the main page in order to reach the one that does the batch geocoding is difficult, requiring the user to focus on an information dense area with potentially misleading names.

Input is inserted through a simple text area, which accepts one address per line. This is a case where usability can only be measured on the ability to copy and paste, and since that is something handled by the browser, not the application, it is not valid for this study.

The Geocoding process received a Neutral rating. The process can be monitored, every time an address is geocoded, the results are always displayed and are immediately available for use. The whole process is susceptible to network failures, but when they happen the results are not lost. Resuming the process is realized manually and requires the users to check themselves where the process failed and resuming from there.

The output step also received a Good rating. Geocoding results come with a confidence level with a scale of 0-9 by region, 0 being a failed geocoding, 1 being a country and 9 an exact building. These results are available within the application and limited to the session. If users close the page, the results are lost. They are easy to export, coming in textual CSV format.

2.1.3.3 Batchgeo

This application's interface received a Good rating. It gives the user a pleasant and fluid experience with direct and simple instructions. There are examples of the kind of input data the application expects and once users insert their own it is validated and can be verified.

The input step also earns a Good rating. Users can tell the application what is kept in which fields and there are examples both of what is expected and of the outcome of the process. Extraneous information is safely ignored by the application and does not impede the geocoding process.

The Geocoding process, once again, receives a Good rating. The progress can be monitored and is cancelable. When the operation is cancelled, the user still has access to the geocoded information, and can even continue to the output step with it. However, it does not tolerate network failures. When they do happen, it halts, and although the processed information is not lost, the steps to salvage it are not intuitive; it is also impossible to resume it without starting over.

Even though the output options are very limited, the one they do provide is very well designed, which gives this application a Good rating. Results are presented on a map enabling manual correction and the generated data is permanently stored within the application. Even though it no confidence indication is provided, verification of small sets of results are facilitated by the map visualization that the application provides.

2.1.3.4 GPS Visualizer

The Interface for this application evaluated to a Bad rating. Instructions are undifferentiated from main text and, once found, are unclear and hard to read. Input fields are disorganized and, aside from the label, no explanations for their purpose is provided.

As with Find Latitude and Longitude the input step on this application is handled by the browsers ability to copy and paste the information, and as such it can not be evaluated.

The Geocoding process earned the only Good rating of this application. It can be monitored and allows its cancellation, although not its resuming. It is susceptible to network failures and the process halts as if users had pressed the cancel button. Every time an address is geocoded the information is immediately shown to the user, both in text and on a map. Validation of information can be done using this map.

The output step comes back down again to a Bad rating. The only two factors in favour are that results are accessible within the application and are easy to export as textual CSV data, a geographic file type or as an image. This is, however, not enough to overcome the fact that data verification is hard for more than a small number of requests. Results are temporary and closing the site will void any work done.

APPLICATION	RESULTS	FEATURES	USABILITY
Bulk Geocoder	-	·	·
Find Latitude and Longitude	-	·	·
Batchgeo	-	-	+
GPS Visualizer	-	-	·
Topo.ly	-	·	+

Table 2.4: Usability scores of the researched applications

2.1.3.5 Topo.ly

This application earned a Good rating on the Interface. Besides the fact that is has a few interface elements that are not intuitive and may confuse the user, it is well designed. The overall experience is smooth and pleasant, its use is easy and responsive, and there are clear and contextualized instructions on every step of the process.

The input step gained a Neutral rating. It accepts extraneous information on input, saving the user the trouble of its manual removal. It has a validation step but it is not user friendly. Most error messages are too cryptic for normal users, leaving them confused and without alternatives to proceed. To know which types of header the application expects, users need to leave the process and look for a solution in the FAQ page.

Failures in the Geocoding process bring it down to a Bad rating. The process can be monitored and cancelled, but when users cancel the process all data gathered until then is lost. The application has no way to recover from problems in the network and any of those problems voids the operations. There is an indicator showing the progress of the process, but no partial results are given.

In the output step it comes back to a Good rating once again. The visualization of data is highly simplified by using a map, the generated data is permanently stored, and results are accessible within the application without requiring its download. Unfortunately, we could not gather any information on file downloading, since every download option is blocked until payments are provided. Additionally, no confidence indicators of any kind are provided.

2.1.3.6 Analysis Conclusion

The overall evaluation of the five applications can be found in Table 2.4. As mentioned before, every application performs poorly when geocoding edge cases addresses, position most addresses in the wrong locations. In itself, failing to geocode is not necessarily bad, what brings the score down is that all the applications geocode almost every address on the wrong locations, leading the users into erroneously believing they got accurate results. It is not wrong to fail, but it is wrong to give incorrect results.

On the features front, most applications provide at least some functionality. Bulk Geocoder and Find Latitude and Longitude accompany their results with confidence levels, which, in theory, would be a great feature, but in practice, because most results come

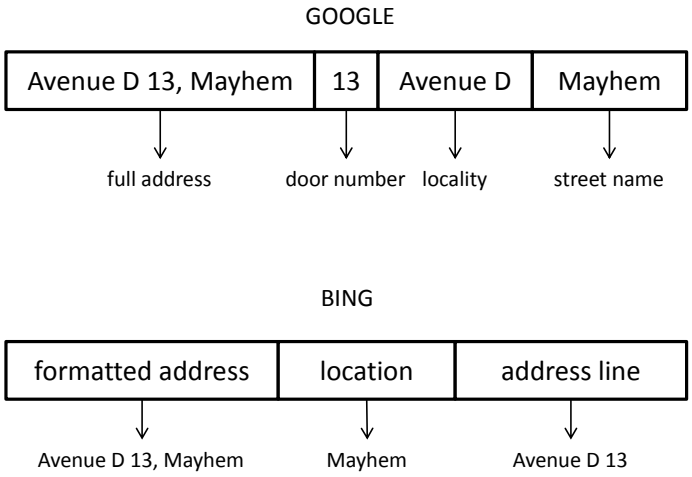

Figure 2.1: Street components information

out wrong but with good confidence levels, only further misinforms users. Batchgeo and Topo.ly do not provide any confidence levels with their results, but compensate by allowing their cartographic visualization. This is an ad-hoc verification mechanism, and while is not as usable as confidence values, in this particular case, where most results come out wrong, it provides more accurate information to users. The lack of ability to really give useful and true confidence metrics is what stops the applications from being given higher scores.

While none of the applications have particular detrimental usability, most of them could be improved. The exception being Batchgeo, which earned positive evaluations on every front. The one downside of the application, in comparison with others, is the lack of access to the generated data, but for what the application was designed to do, its usability is an example to follow.

2.2 API Comparison

A requirement for this project was the detailed comparison of two different geocoding APIs: Google Geocoding API and Bing Maps Geocoding Service. Along with the examination of the documentation of both [Bin13, Goo13], they were also tested to obtain a more deep knowledge on their inner workings and ease of use.

When they are not certain to provide a correct result, both services will answer with more than one set of coordinates. One of them will normally fit the query better than the others and will come in first place in the ordering of the results.

As can be seen on Figure 2.1, both Google and Bing will provide information on the type of result (street, road, locality, etc) both for the full address and for each address component, but the data comes organized in a slightly different way.

While in Bing we have direct access to each component of the address, in Google developers are forced to search every component until they find the one they are looking

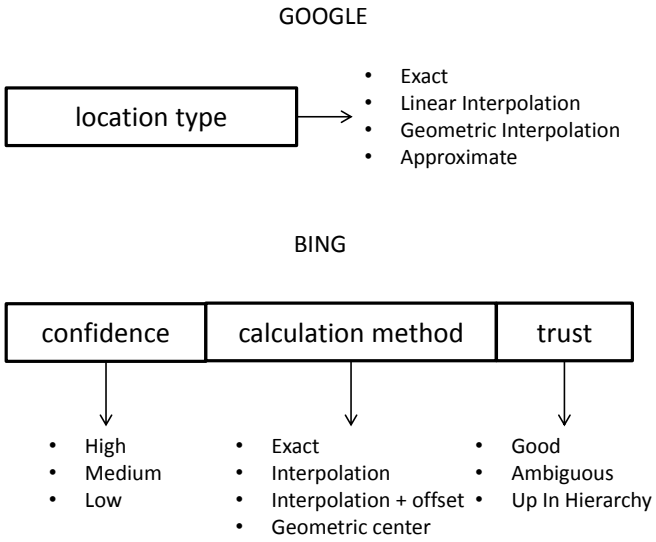

Figure 2.2: Different types of confidence information for each service and their values

for. This comes at the cost of future flexibility. If a new type of address is created, Google does not need to change it's data structure to accommodate for it, while Bing will have to introduce a new version of the API, which can represent it, or be forced to use a non-standard field.

Both services have a field indicating how the value was calculated. Google only differentiates between *exact*, *linear interpolation* and *geometric interpolation*. Results which do not fit into this category will appear as *approximate*. Bing has one more option, *linear interpolation with offset*, but the most inaccurate option is *geometric interpolation*, which means the worse results will all gather under this value.

Bing comes with two more fields which can be used to infer confidence and accuracy on the results. The first one is a field called *confidence*, which can have the values of "Low", "Medium" or "High". The other, *trust*, has three possible values: *Good*, meaning the result has a good quality; *Ambiguous*, meaning there is more than one possible result for the address; and *Up In Hierarchy*, which indicates that the service was not able to find a result for the more precise address and instead returned the one above it (for example, returning the position of the location (e.g. the town) instead of the street). To have a clearer, less-textual idea of what each service has, check Figure 2.2.

Candidate Results are the ones that also seem to fit the query, but in which the service has less confidence than in their first result. They are present in case the application using the service is able to recognize any of them as a more fitting alternative. Figure 2.3 shows that Google has much more information for these. It answers with full information on them, while Bing will only give information on coordinates and calculation method. This means that each Bing response will be a lesser strain on the network. However, if more information is required by the client, the strain on the network will increase because new

GOOGLE

Best Match	Candidate Match	Candidate Match	...
Full Information	Full Information	Full Information	...

BING

Best Match	Candidate Match	Candidate Match	...
Full Information	Coordinates + Calculation Method	Coordinates + Calculation Method	...

Figure 2.3: Information given for each candidate result in both APIs

requests will be made, while with Google there is no such need, since all the information comes in the first response.

Not as important for the results themselves, but still interesting while displaying them in a map, is that both services answer with a bounding box. This box has coordinates which make up a rectangle, which can be used as an appropriate viewport when focusing on the cartographic representation of a single result.

2.2.1 Google Geocoding Service API Analysis of Results

Becuase Novabase required the use of Google Geocoding Service API, a more thorough analysis was performed on this service's results. The second sample file mentioned in 2.1.1, which had the more difficult addresses, had its results carefully analysed. The results are presented in this section.

The minimum accepted distance from a real location to a result given by a geocoding service varies with the intended use of the information, but for the purposes of these analysis, twenty five metres will be considered the maximum distance at which a location is considered correct. A larger difference will be interpreted as a geocoding failure.

Analysing Figure 2.4 it is possible to see that only 20% of the results fall within twenty five meters of the correct location. The remaining 80% are geocoded far away, rendering them erroneously located. If one assumes that results which are more than one kilometre away from the correct location are so because the address has been incorrectly parsed (and as such, can be corrected), there are still 50% of addresses which are positioned at a distance of more than twenty five metres from the real location.

Despite that, one could argue that Google results come with a confidence result associated, which can be used to avoid accepting wrong results as correct. Further analysis, shown in Figure 2.5 shows the confidence values which accompany Google results.

This analysis is somewhat skewed, since the data was purposely selected such that Google would fail many of the results. By removing the failed results it is possible to get

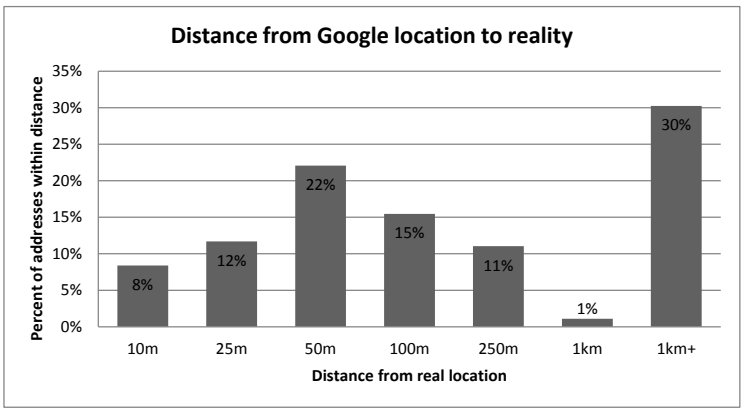

Figure 2.4: Difference in distances from Google Geocoding Service coordinates to the real locations

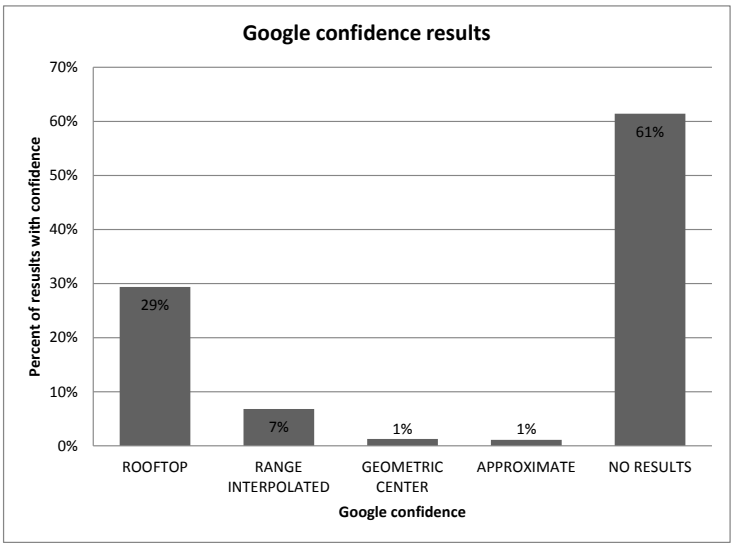

Figure 2.5: Variance of Google Geocoding Service's confidence on its results

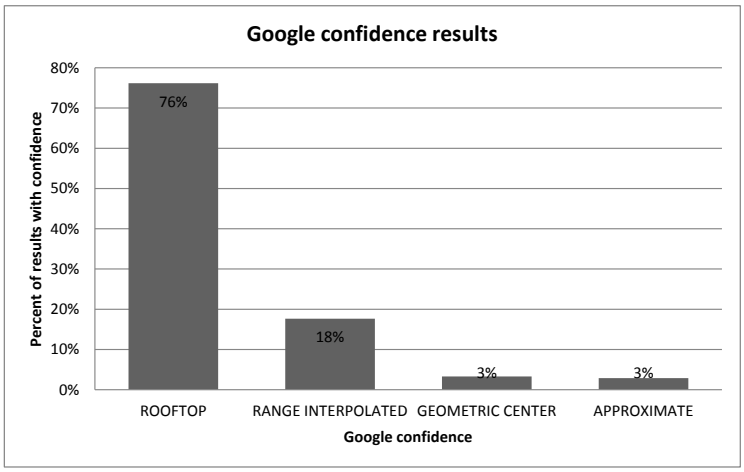

Figure 2.6: Variance of Google Geocoding Service's confidence on its results (not including failed results)

a better idea of the results obtained. The modified graph is in Figure 2.6.

From the graph it is possible to infer that in 76% of the results, the confidence value provided is the highest provided by the service. This was an unexpected result. The data being used for testing was chosen so that Google would have trouble geocoding the addresses, a behaviour which can be observed in the graph which does not omit the failed results (Figure 2.6). It was, thus, to be expected that only a very small part of successful results would be accompanied of a good confidence value.

To better understand this behaviour further analysis was done on the obtained data. For each confidence level the distances from the results to the real locations was gathered. The most important results are displayed on the graph of Figure 2.7.

Only 27% of the results are within the bounds of a valid location (an area with a radius of twenty five metres). The remaining 73% results are displaced from the true location by more than twenty five metres. Nonetheless, Google erroneously attributes them the best confidence value they attribute to any result that they generate, the `ROOFTOP` confidence level, which indicates that the result is in the exact location of the address.

From these results one can now understand how the service gives a 76% best-accuracy rate for areas which were explicitly selected to target its lack of accuracy. The Google Geocoding Service does indeed have a low accuracy on the areas, but it reports its results with high accuracy.

If one trusts only the best results from Google and disregards results which do not come with the best confidence level, `ROOFTOP`, then 29% of all given addresses passed onto it will have at least one result. Of those 29%, only 27% will be within the twenty five

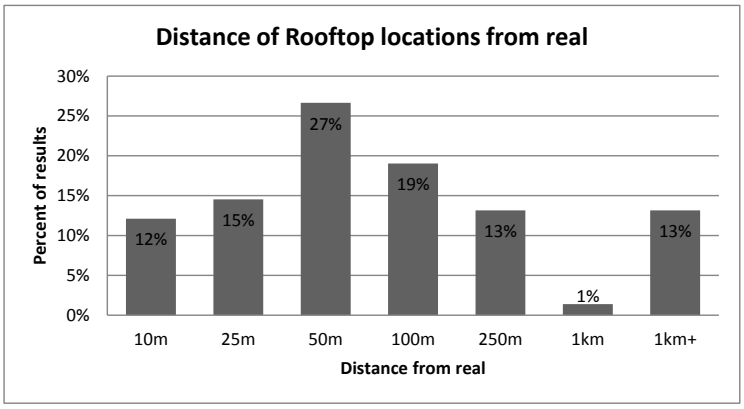

Figure 2.7: Difference in distances with ROOFTOP confidence

metre radius which we ascertained would be the area for a correct result. This means that of all the addresses passed onto the service, only 7,83% of them will carry an acceptable correct codification for the addresses. Taking into account that 71% of the addresses give no valid results, we end up with a total of 21,17% of the total addresses being erroneously geocoded.

Extending the previous analysis to also include results with confidence levels besides ROOFTOP leads to another interesting development. As can be seen in Figure 2.8, if one takes into account all valid results, there will be an almost threefold increase in the accuracy of the results, which means that some results for which the service gives less than perfect accuracy do end up being correct. This leads us to the conclusion that the confidence is not correct, and that it should not be trusted.

2.3 Discussion

From the first study, Application Analysis, we concluded that all applications are certainly usable if one assumes the data does not contain problematic addresses. However, it is not easy for users to distinguish between addresses which the applications are expecting and which they are not, services are expected to answer all queries with the same quality, but this does not happen. The fact that applications do not provide any means for users to ascertain their trust on the results only worsens this, making most applications unusable at organizational levels.

Geocoding applications do not yet fill the target market we are aiming for, they do not allow organizations to rely and improve on the applications results. Additionally,

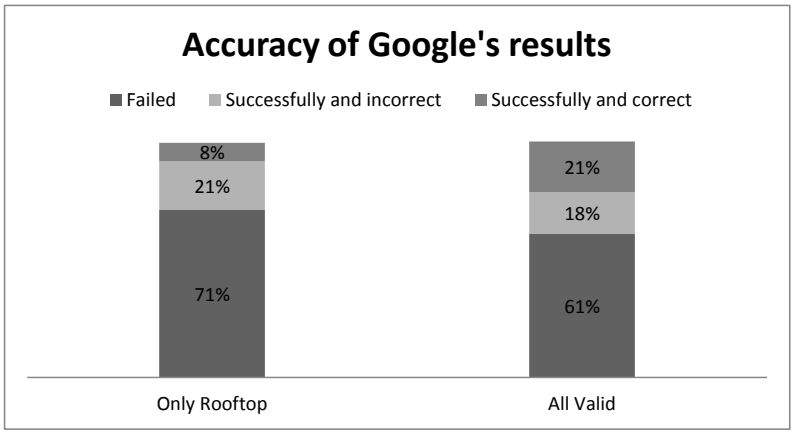

Figure 2.8: Quality of Google's results against real locations

support for confidence values is still rudimentary, there is little variety and flexibility on service providers and ease of use is still constrained. All these are factors which keep users from using these applications and their geocoding technologies.

Our proposed solution helps solve these problems in two different ways. The first is by allowing the insertion of a knowledge base which is capable of augmenting the overall quality of the results, and the second is the ability the final application provides of allowing its users to correct the results obtained by the geocoding process.

As for the second study, API Comparison, it revealed that none of the APIs have an overall superiority over the other. Bing has more information on confidence results, but the added information, when compared with Google, is not enough to be of significance. Google responses come with more information on candidate addresses, but this is the standard result information which the user may not need, resulting only in wasted bandwidth.

In the end, the choice was made not from a perspective of the best technology but from a perspective of context and familiarity. This was a strategic choice by Novabase, the rationale being that as Google Maps are more widely used and recognized, it makes sense to choose the technology which most resembles what is already being used.

The further studies done on the results of the Google API revealed to be very interesting. The failure of the confidence levels was unexpected and if not resolved would be a serious problem for users' trust in our proposed solution. However, the feedback module included in the final application handles this problem well. By voiding the trust in confidence levels and doing its own analytical analyses it provides users with the necessary hints and warnings which are a better fit for indicating confidence on the results.

3

Methodology

Geocoding is a complex process. Data, such as addresses and geographic coordinates, need to be collected in the field to be processed and inserted into a knowledge base. The address component of the data is not uniform, different countries have different rules for describing an address, and even a single country can have more than one way do to it.

To better understand this problematic and obtain insights which could be useful to the project, we took the initiative of collecting data in the field.

There was one type of address which was known not be handled correctly in mainstream geocoding services, due to it having a specific format, which breaks current rules of door numbering in Portugal. Further testing on some of those services, such as Google and Bing Maps, revealed that this was indeed the case. Of the affected zones we choose two in *Parque das Nações, Lisboa* and visited them to collect the correct addresses and coordinates.

To collect the data, aerial images of the chosen zones were printed to take to the field. Once there, we would highlight in each sheet of paper each of the addresses found, and note down the street and door number. The compilation of addresses resulted in around three hundred addresses collected over the span of one week. The addresses were then georeferenced and converted into digital format using a GIS application. The locations of the addresses were plotted onto a digital aerial map, and from those the application was able to retrieve the geographic coordinates for each location.

The knowledge base now has three hundred addresses, but these only cover one very specific area, in which it was feasible to gather data via field work. To widen the spectrum of our data we used geographic information on street addresses provided by the *Câmara Municipal da Amadora*. The data was studied and the addresses which failed to be correctly geocoded by mainstream geocoding services were selected to be included in

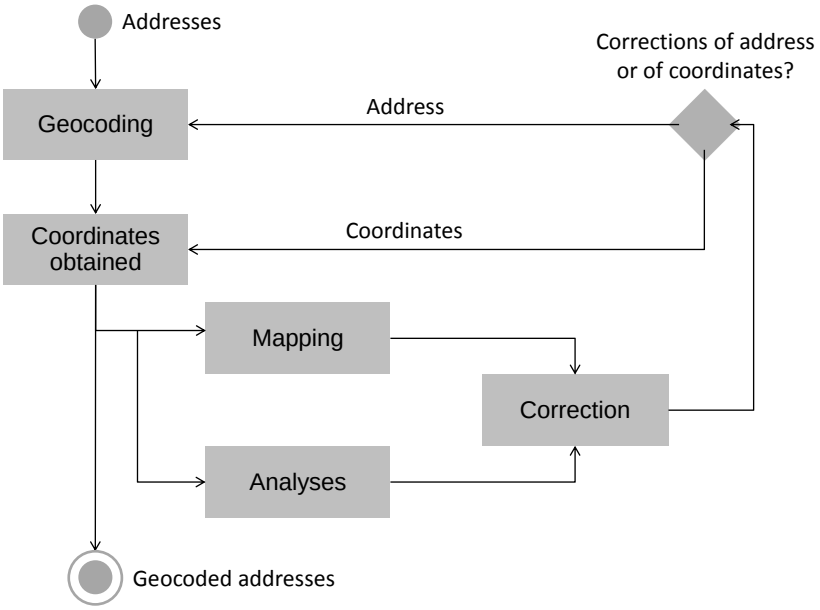

Figure 3.1: Work flow of our proposed solution

our knowledge base.

With these two sources loaded into our knowledge base, the number of addresses available rose to eleven hundred. With these addresses it became possible to test mainstream geocoding services against real data. Of those available Google Geocoding Service was the chosen one.

Taking into account the previously established objectives and this information obtained through the studies done in State of the Art, we devised the work flow shown in Figure 3.1. It is important to note that the knowledge base is not meant to cover every single address it may be thrown at it. It should be thought of as an improvement on the quality of the results of other geocoding services, to be used side by side with them to obtain the best quality data available.

However, to obtain the most accurate geographic coordinates, Novabase clients must provide a series of addresses to build the knowledge base for the application. The final product is still able to provide functionality even without it, but this knowledge base is the one with the potential to greatly increase the overall quality of the results.

The work flow is composed of six distinct steps:

1. **Addresses** The provision of addresses to the application

2. **Geocoding** The process by which the application obtains coordinates for all the addresses

3. **Coordinates obtained** The coordinates are saved and made available

4. **Mapping** The addresses are mapped to the coordinates obtained in step 2
5. **Analysis** A series of operations are effectuated in the obtained results to generate feedback to users
6. **Correction** Users can chose to correct any number of locations, depending on the nature of the correction these are inserted back in steps 2 or 3
7. **Geocoded addresses** Users obtain the geocoded addresses

In step 1, to obtain the most accurate geographic coordinates, users (Novabase client companies) must be able to provide a series of addresses. The more these addresses match with the ones in the knowledge base, the best the results will be. The individual components of the addresses (street name, door number, etc.) can be provided independently from each other or joined together in a single unit of information. In each case, the application tries to distinguish which pieces of information are relevant and can be used to find the address coordinates, ignoring the rest.

In step 2 these addresses will then be geocoded against the application internal knowledge base. This knowledge base is always considered of utmost confidence, and results from it take precedence over any others. The maintenance of the knowledge base is the responsibility of the ones responsible for the system. When the information in the knowledge base is insufficient to find the coordinates for an address, we fall back to using other geocoding services to find them.

Every pair of coordinates is accompanied by a confidence level in textual form. There is one confidence level for each corresponding degree of confidence the geocoding service gives, labelled in common language which replaces the GIS terms the services normally use. Aside from those, one label is reserved to denote results coming from the knowledge base of the system, in which we have the greatest confidence.

The number of gecoding services available for geocoding can vary, depending on the configuration of the system. By default the system only uses on geocoder, but organizations can configure it differently to use more than one, and create a logic which dictates their precedence.

This way, the process can be thought of as contribution to the improvement of the geocoding results of the current geocoding services. In the worst-case scenario, our results will be the same as if users had consulted the mainstream geocoding services, but with the right knowledge base to back it up the results will be improved through the use of our system.

In step 3, the geographic coordinates and their confidence level are shown to the users and made available for download. The only information that is required to be provided by the geocoding process are the coordinates and the confidence level associated with the given original address. However, if possible, the original information will be maintained, and the new information will be made available along side it. The presence of the original information is constrained by the file format in which it is requested. Some file formats,

such as CSV, allow for arbitrary data, but others, such as GML, do not, and in those cases the original data will not be present in the resulting files.

In the 4th step, the results are visually exhibited in a map. This gives users a cartographic context, enabling ad-hoc spacial analysis of the results. Some kind of errors are easy to detect using this visualization of the data, but some others are much harder to detect, and require other kinds of technologies.

This is were step 5 comes in. The results are analysed by the application, which will try to find errors and strange results which may also be wrong. It is important to note that these analyses do not guarantee that every resolve found to be strange is necessarily wrong. These analyses are intended to be a hinting mechanism which helps users identify problems in the results. It is possible that the analyses generate false positives, which is to say that it can mark a result as being wrong when it is right.

In step 6 users can then, if they wish, correct results. Once again, it is not necessary to note every warning generated by the analyses in step 5. Users chose which results they wish to correct, if any, and these are sent back to be corrected within the application.

Depending on the kind of correction, the work flow loops to a different point in the system. If users correct the address, this new address will go back to step 2, Geocoding, where it will once again be geocoded. If users correct the coordinates, there is no need to geocode the address again (and no point, since geocoding the same address would give the same coordinates, which are being corrected), instead they go immediately to step 3, where they will directly override the previous obtained coordinates. Both kind of corrections will then unite on the 3rd step, and cycle will be repeated until users decide not to apply more corrections.

In the 7th and final step, users are able to access the geocoding results. These results will contain not only the given addresses, but also the geographic coordinates (latitude and longitude) of each address, plus a confidence level which denotes the confidence the application has on that particular result. As mentioned before, confidence will be a maximum for results which come from the knowledge base. As for the others, they will have the confidence level given by the geocoding service which generated them.

4

Implementation

Many choices had to be made in regards of which technologies to use. We did not have the time to analyse every piece of technology to decide which ones would be the best fit, and because of this most choices ended up being more a matter of familiarity than of performance. The logic being that, considering the time frame to limited, it would be a better choice to use technologies with which we are familiar and can more easily and efficiently used, than some more powerful technology which would have to be learned, thus depriving us much resources in terms of allotted time.

For the implementation of the methodology described in the previous chapter, a client-server approach was chosen, detailed in Figure 4.1. The figure also shows the structure of the proposed solution. The server is implemented in the Java language, version 7, using Apache Tomcat version 7 and connected to PostgreSQL Database version 9.2.

The client is implemented using Web Technologies, namely, it uses the Angular.js framework which assists the development of single-page applications and augments browser-based applications with MVC, and Google Maps Javascript API to enable the use of maps and help the visualization of results. Initially we focused both on functionality and browser compatibility, but as time became a scarce resource it became a reality that there was not enough to focus on the two, which led to browser compatibility being dropped in favour of functionality.

The application flow starts in the client, where users can specify one file containing one or more addresses and send it to server.

On the server, the addresses contained in the file will be geocoded and a new file will be generated. In this new file, besides all the information contained in the original file, three new pieces of information will also be included: latitude, longitude and confidence, one set of each for each address found on the original file. In addition, every time an

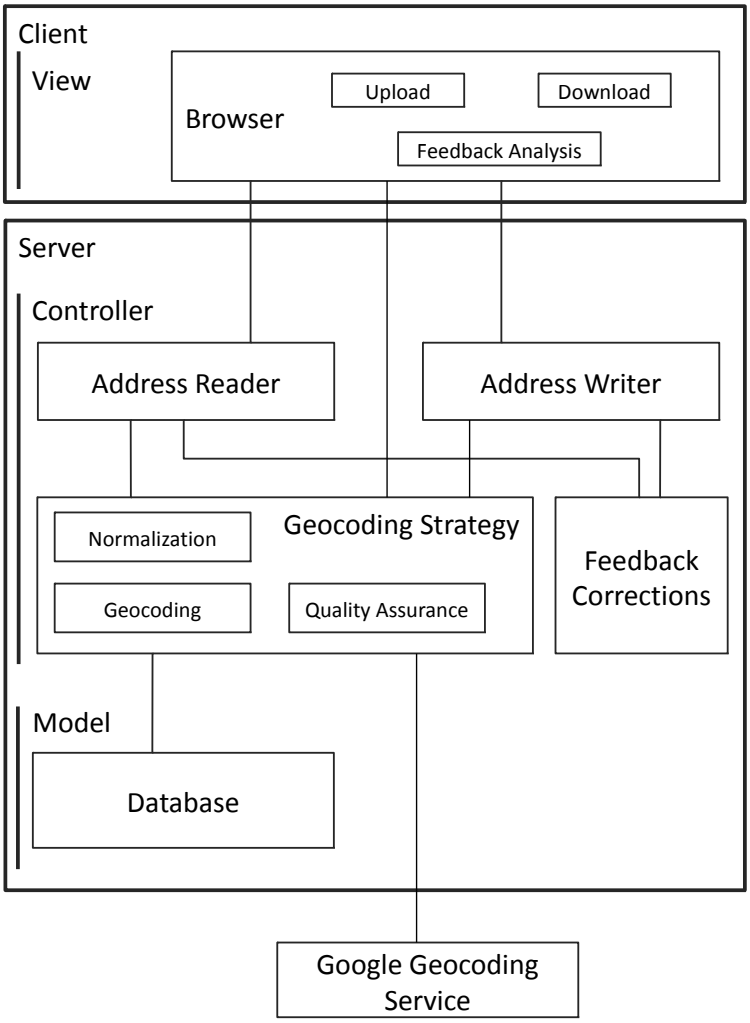

Figure 4.1: Application components and flow

address is geocoded, the corresponding information will be sent to the client, so it can be updated in real time.

The information sent to the client will be shown in both table and map form. The table form will include the fields of the original file, plus the three added by the geocoding process and one more which will contain the feedback generated by analyses executed once all the information is available. The map form makes use of the Google Maps Javascript API version 3 to display every geocoded address as a point in the map, and enables a cartographic view of the generated information.

Once all addresses have been geocoded in the server, a new file is written and made available for users to download. At the same time, once the client application has access to all the geocoded addresses, it will perform a series of analysis on them. These analysis will try to find possible problems in the results and warn users about them.

Users will then have the option to correct the results obtained. These corrections will be sent back to the server, which will rewrite the results file with the corrected information, allowing users to download the file with the corrections.

4.1 Data Entry

This is the process of feeding addresses to the application through files. In this Section we will talk about how the process is handled on the client, how users select their files and how they are sent to the server, and how it is handled by the server, how it receives the files and how they are saved to disk for access by other steps of the geocoding process.

More than describing how the file reaches the server, we will also describe the process by which it is read to an in-memory representation. We named this internal representation `InformationUnit`. A series of operations can then be applied to this data structure, and shared between components of the application.

The addresses to be geocoded are supplied to the system using files. Through the client, users can upload a file to the server, where it will be saved to disk.

The file is uploaded using Javascript XMLHttpRequest Level 2 with FormData. The FormData object holds key-value pairs of information, and once users choose a file, we load this file into the FormData object and append it to the XMLHttpRequest before sending it to the server.

The advantages of using the XMLHttpRequest Level 2 are twofold: 1. there is no need to reload the page, since the request can be asynchronous, and 2. it allows easy implementation of progress notifications, since it allows us to listen to upload progress completion values.

Saving to disk in the server side has both advantages and disadvantages. One clear disadvantage is that it takes up disk space and requires file system management. However, taking into account the advantages, these are no reason to not take this approach. One of the advantages is that it allows the resuming of operations in case of server problems. Other is the possible implementation of queueing strategies to help reduce server

load on peak times without increasing the burden on the memory used.

The advantages mentioned were not implemented, since they were deemed unnecessary for the purposes of this project. However, the way is paved so that implementing them requires minimum effort.

When saving files to disk it may occur that two files with the same name will be uploaded, and the last one will override the other. To avoid this, the server first checks if a file with that same name already exists, and while it does exist, it will try to create a file with a number between parentheses appended to it. For example, if the user uploads a file with the name `filename` and it already exists, the server will try `filename (2)`, `filename (3)`, etc. until an available name is found. After the file is successfully saved to disk, the name with which it was saved is sent back to the client application.

Now that the client has the file name, it needs to open a WebSocket connection to the server and send a message to continue the process. The message sent has the format `geocode <filename>`, where `filename` denotes the name of the file uploaded. This is not the name with which it was sent, but the one returned by the server.

The reason for this intermediate step has to do with the aforementioned reason for saving files to disk, to allow implementation of queueing strategies. Creating the connection and sending the message allows the server to, for example, reply back with a negative answer, indicating with it the expected delay until the file can be geocoded, which the client can wait for before retrying.

As soon as the file is saved, and the geocoding request is received through the WebSocket connection, we proceed to reading it. The contents of the file are translated into an in-memory representation with the objective of abstracting file-type specifics. This abstraction, which we called `InformationUnit`, simplifies code. With it in place only file readers and file writers need to be aware of file types, every other part of the system simply deals with this data structure.

This structure is an abstraction for an ordered list of information units read from the file. An information unit is a single unit of information which makes sense on its own. For the case of a CSV, an information unit would be a row, for the cases of KML and GML, it would be a node. Each information unit is accompanied with an integer that denotes the order by which it was read. The first unit read will have the number 1, the second the number 2, and so on. This is useful to preserve data order when writing the results file and essential for the corrections which will be described in the following section. Also worth mentioning is that this structure only exists in-between the reading and writing of a file. Once a file is written, the in-memory representation is discarded, and to re-acquire it is necessary to read the file once again.

There are two interfaces which declare how these information units should be read and written from file. A Java class implementing the `InformationExtractor` interface can be associated with a file type and loaded into the system to read the information from files. In the same way, a Java class implementing the `InformationDispatcher` interface can be associated with a file type and loaded into the system to write information to

files.

However, reading and understanding the contents of a file takes substantial effort. The way files are written and how the data within them is organized can vary even between files of the same type.

Files written in CSV are one example of this. Even though the specification [CSV13] says that values must be separated by a comma, Microsoft Excel uses a semicolon. To allow this kind of deviations, we chose to require a configuration file for every file type the system supports.

Along with the information on how to correctly parse and read the file, file configurations are allowed to have extra information. Once again using CSV files as example, the data is normally distributed among various columns in no particular order, which hinders the ability of the system to understand the information contained in the file.

At this point in the process, the information needed from the files is the address to geocode. The configuration files can specify rules which instruct the program on how to correctly read and understand the information contained in the file.

In most cases these rules are very simple. There are six fields which can be specified: street, locality, administrative areas level one, two and three, and country. Aside from the street, every field accepts an integer as access rule, which is to say that it directly corresponds to one field in the information unit. Currently, only access by position is implemented. Access by name was though of, but it was not finalized and is not fully supported. An example of a valid CSV configuration file is in Listing 4.4, in the File Configurations paragraph of Section Server Configuration.

The street field is more complex than the others because it needs both the street name and the door number to be complete. This choice was made based on the expected use of the system. The purpose of our solution is to geocode addresses with accuracy on the house (e.g. the Google `ROOFTOP` classification), and as such it does not make sense to allow a street name without a door number. The other fields are optional to allow omissions on the matching. This way, an organization which deals with addresses in a specific country does not need to add that information to the database or to the files, and is implicitly matched, which is to say that, if neither the source file nor the database have information on (for example) the country, the field will always generate a positive match. However, if one of the two does have that information, then it will be required to be present in both for a match to occur.

Due to this, instead of simply accepting a position, the address field accepts an array of both integers and strings. The integers are indexes for the file columns, starting at 0, and the strings are characters which need to be put before or after the columns of the file for the information to make sense. For example, think of the information in Table 4.1 and the rule `[0, " ", 1, ".", 2, ".", 3, ".", 4, "", 5]`. The results would be:

1. Avenida D. João II 1.3.2.1A
2. Avenida D. João II 1.7.2.1

STREET NAME	NUMBER 1	NUMBER 2	NUMBER 3	NUMBER 4	LETTER
Avenida D. João II	1	3	2	1	A
Avenida D. João II	1	7	2	1	
Avenida D. João II	1	16	5		F

Table 4.1: Example CSV file with addresses

3. Avenida D. João II 1.16.5.F

4.2 Geocoding Process

In the context of this application and in order to have a quality result, geocoding of one address my need to be performed more than once. This need will depend on the configurations loaded when the server was started. By default, each address will be geocoded one to two times.

The first geocoding attempt is done against the internal database of the system. If one and only match is found the geocoding process of the address terminates here and the answer is saved for posterior writing.

The second geocoding attempt is executed if and only if the first attempt did not end with a valid result. In such case a request is sent to the Google Geocoding Service. The results can be composed by one or more pairs of coordinates and, if there is only one, that is chosen as a valid result, if more than one are present, the first one is chosen. When the service does not return any result, or if for some reason we are unable to access the service or its results, the geocoding of the address fails, and no coordinates nor confidence value will be provided for the address.

Every time an address geocodes successfully, the coordinates and confidence results are saved together with the information obtained from the file, in the respective information unit, as additional fields.

Geocoding huge files of addresses is a time consuming task. To help give the user a sense of progress, every time an address is geocoded, more than saving the information for posterior writing, we will also immediately send this information over to the client application through a WebSocket connection, which was previously created by the client. The client application will then be able to display and act upon it.

The message follows the format `geocoded <json object>`, where `json object` is the json representation of the whole information unit pertaining to the geocoded address, plus four additional fields:

- **$_address** The formatted, textual address which is used to refer to the associated location;
- **$_latitude** The latitude coordinate of the geocoded address;
- **$_longitude** The longitude coordinate of the geocoded address;

- **$_confidence** The confidence associated with the result.

4.2.1 Geocoding with the Database

As mentioned in the Introduction, before an address can be matched to its coordinates, it needs to go through the normalization step.

Implementing an address normalizer is nowhere near trivial, a whole thesis could be written on the subject. As such we did not attempt to write one, instead the system makes available extensibility hooks (see Section 4.5, Extensibility) which enable system administrators to provide other normalizers.

We planned on using an address normalizer which is being developed in the organization. However, the state of this normalizer at the time of writing is yet far from usable for our purposes, and as such it is not yet included with the system.

With this in mind we did implement a very simple address normalizer which is bundled with the system. This normalizer is in no way complete and is not recommended for production use, but its inclusion was necessary, both for proving that the normalizer hook is indeed working as intended and to enable a wider range of tests of the system as a whole.

Our very simple normalizer executes five operations on any given addresses:

- Removal of double white space characters;
- Converts every letter to its lower-case counterpart;
- Replaces single letters (technically, any characters) with any other combination of letters, as defined by the file in `config/replacement_characters.js`;
- Replaces words as defined by the file in `config/replacement_words`;
- Removes stop words as defined by the file in `config/stopwords.js`.

The character replacement step is used to replace some letters with others, for example, reducing many letters with diacritics to their common non-diacritic root.

Replacement of words is useful for replacing full words with their equivalent forms, such as changing `St.` to `Street`.

Stopwords are words which do not add meaning to a phrase or address, such as conjunctions (and) or prepositions (of, in, on). As these appear in many addresses, and are sometimes omitted, they are removed from normalized addresses.

For a more concrete example of the execution of these operations, take a look at Figure 4.2, where the address `R Pedro e Inês 17` is normalized to `rua pedro ines 17`.

With the address and its components now normalized, it is now possible to match the address to the information on the database. The database is structured in three tables, as shown in Figure 4.3.

original	R Pedro e Inês 17
to lower case	r pedro e inês 17
character replacement	r pedro e ines 17
word replacement	rua pedro e ines 17
word removal	rua pedro ines 17

Figure 4.2: Normalization example of a simple address

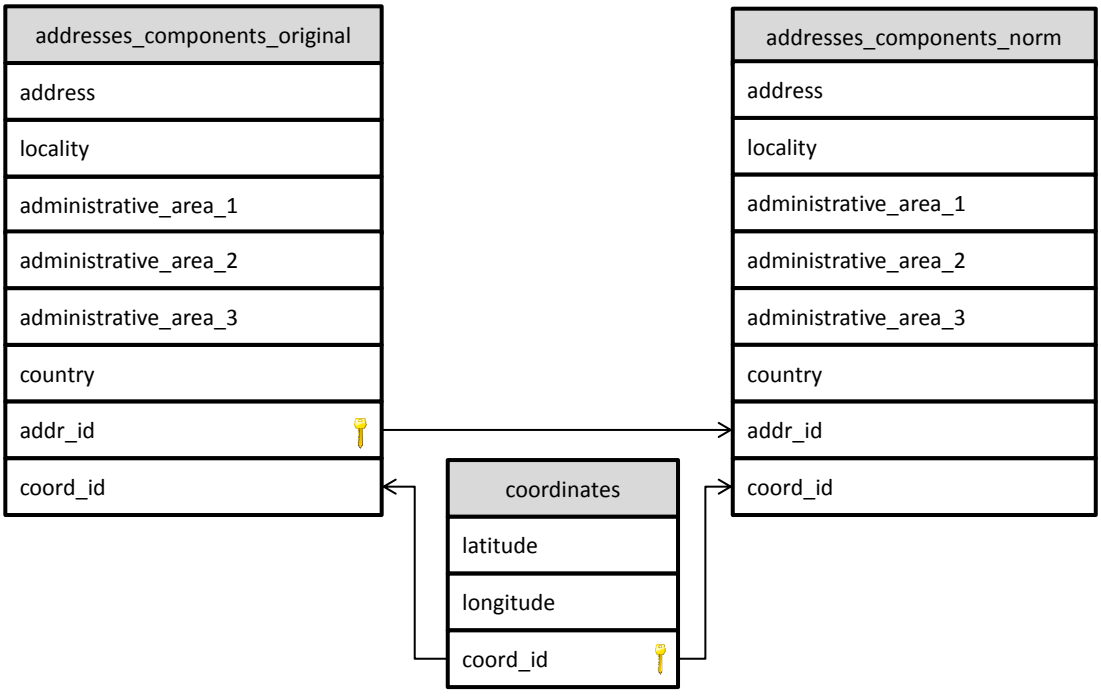

Figure 4.3: Table structure of the database

The `coordinates` table contains the latitude and longitude pairs for every address in the database. The `coord_id` column is the primary key.

The `addresses_components_original` table contains the original addresses given to the database, these are not normalized, and will be necessary if the normalizer for the system is ever changed. The `addr_id` column is the primary key, and the `coord_id` column is a foreign key which references the column `coord_id` of the `coordinates` table.

The `addresses_components_norm` table contains the same columns as the previous table, but unlike it, the every address field is normalized according to the address normalizer currently in use. The `addr_id` column is a foreign key which references the column `addr_id` of the `addresses_components_original`, and the `coord_id` column is another foreign key which references the column `coord_id` of the `coordinates` table.

The need for two tables with almost the same data stems from the fact that addresses need to be normalized. If it were not included, the `addresses_components_original` table would suffice, but since it is, we need the `addresses_components_norm` table to hold the normalized form of all addresses.

Most of the time, only the table with the normalized addresses will be consulted, and in fact, it is the only one needed for the proper functioning of the system, which raises the question of why should we maintain the table with the original addresses. As will be mentioned in the Extensibility section, normalizers can be changed in between server restarts, if the administrator so desires, which raises some problems with this approach.

When the normalizer changes, the results of the normalization of addresses will also change. However, the information on the database does not change to accommodate these, and it will stop providing results, since the addresses normalized with the new normalizer will not match most or all the addresses normalized with the old normalizer.

There is, then, a need to re-normalize the addresses in the database. This is impossible to do if done with the information taken from the normalized table, since in that table, original information may, and most probably will, be lost. For this re-normalization to be possible, we need access to the original information, not modified by the previous normalization process, which is why the original table is present.

The query used to search the database is shown in Listing 4.1, in JDBC syntax, meaning that the interrogation points are arguments to be set at runtime.

Listing 4.1: SQL query for address matching

```
1 SELECT
2   o.address, o.locality, o.administrative_area_1, o.administrative_area_2,
3   o.administrative_area_3, o.country, c.latitude, c.longitude
4 FROM
5   addresses_components_original as o INNER JOIN
6   addresses_components_norm as n ON(o.addr_id = n.addr_id) INNER JOIN
7   coordinates as c ON(n.coord_id = c.coord_id)
8 WHERE
```

```
9   (n.address = ? OR (n.address IS NULL AND ? IS NULL)) AND
10  (n.locality = ? OR (n.locality IS NULL AND ? IS NULL)) AND
11  (n.administrative_area_1 = ? OR
12    (n.administrative_area_1 IS NULL AND ? IS NULL)) AND
13  (n.administrative_area_2 = ? OR
14    (n.administrative_area_2 IS NULL AND ? IS NULL)) AND
15  (n.administrative_area_3 = ? OR
16    (n.administrative_area_3 IS NULL AND ? IS NULL)) AND
17  (n.country = ? OR (n.country IS NULL AND ? IS NULL))
```

The three tables are joined, the two with the address components using their `addr_id` and the table with the coordinates using `coord_id`. From the resulting table we retrieve the addresses components and the latitude and longitude of the associated coordinates. Addresses components are only selected if all their components match the ones given by the normalization process.

The `WHERE` clauses are more convoluted than expected. The problem here is that, by the SQL standard, comparisons with `NULL`, such as `n.address = NULL`, always return false. What this means in practice is that when one of columns is `NULL` in the database or one of the fields in the address is not present, the comparison will always fail, even if it should otherwise pass. To compare with `NULL` in SQL one should instead use the form `IS NULL`. This creates a problem because now, when one of the fields of the address being matched does not exist, there is the need to generate the query using the is-operator instead of equals-operator, which is impossible to do with Java's SQL Statements without having to rewrite the query.

One solution to this problem, which does not require the runtime modification of the query, is to add a second condition to the where clause. The first condition, `n.address = ?` will match if both sides are non-null and the strings are equal. The second condition, `n.address IS NULL AND ? IS NULL` matches when both the column in the database is `NULL` and the field from the address is missing.

This query returns either one or zero addresses. If a result is retrieved from the database, then it is selected as the correct geocodification of the given address, and given a confidence value of `Novabase` to indicate that it came from the database. When no results are retrieved from the database, this geocoding step fails and execution passes to the next geocoding service, which is the Google Geocoding Service, for a new attempt at geocoding.

4.2.2 Geocoding with the Google Geocoding Service

When geocoding with the database fails we fall back to the Google Geocoding Service. This web service allows us to send an address to Google and get back some information on the address, including latitude, longitude and method of calculation.

The answer can be retrieved in either XML or JSON formats, and since we are already using JSON in other parts of the application, that is the format in which we request our

responses.

Once the response is obtained, we check for the number of results returned. If there are no results, it means the service was unable to find a match or some communication error occurred, in which case the geocoding process ends without any viable solution.

When one match is found, the process ends with it as the valid result, just as it would if it were a database match. However, the Google Geocoding Service may return more than one valid result. When this happens the first one is chosen as the solution, and the geocoding finishes successfully.

The chosen solution will carry the latitude and longitude information as it came in the service response. Confidence information, however, comes in GIS terms, which may be difficult to understand to our target users. As such, these terms are mapped into more friendly values:

1. `ROOFTOP` → Exact;
2. `RANGE_INTERPOLATED` → Street;
3. `GEOMETRIC_CENTER` → Region;
4. `APPROXIMATE` → Rough Approximation.

4.3 Data Output

Once every address has been geocoded we can proceed to writing the file for users to download. The writing process is similar to reading. The information units, updated in the geocoding step, will be fed to instances of `InformationDispatcher`s. These dispatchers will then write the information to the appropriate file.

By definition, the type of the file being written will be the same as the one that was read. This means that if the input file was in the CSV format, the output file will also be in CSV format.

After the file is written and saved to disk, a message is sent to the client application using the previously established WebSocket connection. The message has the format `finish <relative path>`, where `relative path` is the path to the file which contains the geocoded information. The interface will then be updated to allow the download of this file.

The logic of both the server and the client applications are ready to allow the rewriting of the results file into a file of another type. To start the operation the client needs to send a WebSocket message with the format `rewrite <filename>.<extension>` where `filename` is the name of the results file which was made available before and `extension` is the extension of the desired file type.

Upon receiving that message the server will build a list of files which contain the results of the geocoding process and have the same name. This is easily done because geocoded files are all stored within the same folder, and kept apart from non-geocoded

files, which are stored in another folder. Building the list is a matter of looking into the directory with the geocoded files and look for files with the same name, while ignoring the extension.

Once this list is built, we proceed into trying to read them, which is to say, we will look into every file and try to find a matching `InformationExtractor` which is able to read the file. With the default configuration, it is impossible for this look-up to fail. Because the results file is of the same type as the file originally provided by the user, which we were able to read before, the geocoded file will necessarily have the same file type as the originally uploaded file, and thus be readable.

After we obtain the file containing the results with the whole information and an appropriate `InformationExtractor`, the algorithm proceeds with trying to find an appropriate `InformationDispatcher` for the file type indicated by the extension. If no Dispatcher is found, this process fails and an error message is sent to the client application. Otherwise, the Dispatcher is created and the process continues.

Having a source file, an appropriate Extractor, a target file and an appropriate Dispatcher, writing the new file is a matter of reading all the information units contained in the source file and sending them to the Dispatcher for writing. When the process is complete and the file is saved to disk, the server sends a message to the client informing it of this. this message has the same format as the one sent when the first results file was made available.

We mentioned that an error can occur if the user specifies a file type which the server is not ready to handle. In theory, if the interface were complete, this would not happen. The plan was for, when creating the page, to use JSP to inject into the page the extension the server supports, and the interface only allowing those to be selected. However, due to lack of time, this was not implemented and, in fact, the client application has no access to these file types and cannot request a rewrite of a file through the interface, only through the browser's console. This is one of the features which must be implemented in the future.

4.4 Feedback Analysis

When the geocoded information arrives on the client side of the application, it is immediately displayed both on a table and on a map. Rows in the table can be selected, and this will highlight the corresponding marker which locates the address in the map. Clicking in any marker in the map will bring up a window with the row information associated with it. These visualizations can be thought of as an ad-hoc analysis mechanism, but this experience can also be improved in the future.

Once all the addresses have been geocoded, the client application immediately runs two different analysis: 1. same-location analysis and 2. out-of-cluster analysis. Both analyses are based on our findings of the behaviours of mainstream geocoding services, and are designed by us to handle those specific scenarios. They are explained further in this

section.

The objective of these analyses is to hint to users that, although addresses have been geocoded, there may be some which were assigned incorrect locations. The application does not require that users act on the results of the analyses, even more so because there may be some false-positives. Thus, users are warned about results which *may be* wrong, not about results which *are* certainly wrong.

When the users detect a wrong result, whether it was caught in the analyses or not, they have the option to correct it. This correction can be done in one of two ways.

The first correction method allows the user to move the location assigned to the address to a different one. This is done by making the respective map marker draggable so that the user can place it in the correct location. Once this modification is saved, the new location is preserved in a list of modifications, to be sent to the server when the correction process has ended. This correction can be cancelled at any time during the operation (but not after it is concluded), which re-locates the marker to its original position, before any user movements.

The second correction allows the users to specify a new address to geocode, instead of current one, which will be directly saved to the list of corrections.

Once the users finish the corrections, they can send those back to the server. They are sent to the server through a WebSocket connection, each in its own message. The messages have the following format:

- **Coordinates correction** `correct coordinates <id> <new latitude> <new longitude>;`
- **Address correction** `correct address <id> <new address>`.

The `new` fields carry the corrected information, the `id` field is the identification number given to the information units created when the original user file was read.

After they arrive, the server will read the results file, the one that had previously been made available for users to download, and load the information units from it. This is why the identification number is so important. The order of the information units in the read file must be exactly the same as the one in the written file. If this is not the case, the information units read from the results file, although correct, may appear in a different order, and the corrections will affect the wrong information units, which will result in the new file being wrong.

The re-location correction directly changes the coordinates of the results. The latitude and longitude which were there before will be overwritten with the new ones.

The address corrections, on the other hand, are more complex. They need to trigger new geocoding requests which will be handled exactly as described in the previous section, but with the new address instead of the previous one. As in the previous case, the latitude, longitude and confidence will be overwritten.

After all the changes are done, the old file will be deleted and the new one will be written with the same name, effectively replacing the old file.

Same-Point Analysis One of the common outcomes observed while testing geocoders responses to the difficult set of addresses described in State of the Art was that various addresses would point to the exact same location. It happened commonly when the geocoder recognized the street, or part of it, but not the door number. It would then disregard the door number, and every address which had the same street name would be geocoded to the same location.

This analysis takes hold on that observed fact and that, in the general case, two different addresses are not meant to represent the same physical location, or in other words, that if two different addresses are geocoded to the same exact location, we can assume with a high probability that at least one of the addresses has been erroneously geocoded.

Using that prior information, the previously obtained geographic coordinates are compared to each other, and if any two or more different results have the same coordinates, they are marked with a warning. This warning indicates how many different addresses are located in that same location, and selecting one of those addresses will also select any addresses in that location, so that they are easier to identify in the table view.

The first step is to create a data structure to help us identify these duplicate points. The data structure chosen is made up of Javascript Maps (not a cartographic map). It is a map which maps latitude values to a map which maps longitude values to the geocoded information, or, using a programming example, the following format:

```
\\ {} denotes a Map and [] an Array
\\ x : y denotes a Mapping, where 'x' is the Key and 'y' the Value

{ latitude : { longitude : [ pointer to the data ] } }
```

The first map uses the latitude value of the coordinates as a key. While examining one point, we check for the existence of its latitude as a key in the map. If the key is not present, it will be added with an empty map as its value, if it is, the presence of a map already associated with the key is assumed and we move to step two.

In the second step the longitude of the point is examined. Once again, it may already be present on the map or not. If it is not present, it will be created, and the value will be an array containing a single element: the information associated with the point being analysed. If it is already present, the value associated with the longitude will be changed, the information associated with this point will be added as an element of the previously mentioned array.

In practice, what this means is that we have every unique pair of latitude and longitude as the map and sub-maps, and the values of the sub-maps are the information associated with those points.

After every point is examined and placed in the map, discovering the repeated coordinates is a matter of checking the arrays in the sub-maps. On one hand, if the array has a single element, then it is the only single location at those coordinates, on the other hand, if it has more than one element, all the elements contained in it have the same coordinates, and a warning should be issued for them.

It is important to notice that the elements in the array are not copies of the original information. That information is stored in a separate structure, and the elements in the array will simply point to it, there is no duplication of information in the process.

Out-of-Cluster Analysis This analysis takes hold on extra information known about the data to be uploaded into the knowledge base. In many cases, the locations that are to be geocoded are not isolated points seemingly randomly distributed. More often than not, they will all be located within some bounds. From this, it can be inferred that if some locations are geocoded too far away from the bulk of the locations, then those are, most probably, erroneous results.

This analysis uses that information and tries to locate results which seem to be abnormally distant from others, marking them and warning the user about them, indicating a possible problem with the results.

The algorithm for this one is much heavier and complex than the Same-Point Analysis. First, for every coordinates available we build a bounding box with some predefined length, such that the coordinates point to the center of the box. Then the following algorithm is executed.

The size of the list of bounding boxes is saved for subsequent use. For every bounding box, we check whether it intersects with any other bounding box which follows it in the list. If any two or more intersect then a new bounding box is created, which will contain all of the bounding boxes that were found to intersect. The intersecting, smaller bounding boxes are removed from the list, and the newly created one is added. After the list of bounding boxes has been exhausted, the initial size of the list is compared to the actual size. If the sizes are different it means new and larger bounding boxes were created, which in turn means that new intersections which were not checked may now occur. The algorithm runs once more. If the sizes are the same, nothing has changed, and no more intersections are possible, which ends the process.

Each bounding box maintains information on which coordinates and associated information duel within it. If the number of elements within a bounding box is smaller than some predefined threshold, then the whole collection of elements if regarded as possibly erroneously geocoded, and this is transmitted to users.

Two predefined values were mentioned in the algorithm. One was the length of the bounding box. In the program this quantity needs to be saved as quantities of latitude and longitude, and need to be set manually because at different points in the globe one unit of latitude and longitude will translate to different distances. By default the program uses units of latitude valuing `0.004495` and of longitude valuing `0.005835`, both of which will approximate half a kilometre at 40^{o}.

The second predefined value is the threshold at which a number of isolated points is considered suspicious. For example, if the threshold value was 4, then a bounding box containing four or less locations within it would be marked as possibly erroneously geocoded. However, a single, absolute value for the threshold may prove itself not very

useful. For a file with 1.000.000 addresses, 100 isolated results may be suspicious, but for a file with 300 addresses, it is not. To allow for this kind of adaptation, a third predefined value may be specified, a boolean value which tells the program if the threshold value should be interpreted as a percentage or as an absolute value. If it is set to `false`, the behaviour will be the previously mentioned, but if it is set to `true` the value will be interpreted as a percentage. For example, if the program is set to interpret the value as a percentage, and the threshold value is 1, for a file with 1.000.000 addresses the actual threshold value will be 1.000, and for a file with 300 addresses, it will be 3.

4.5 Extensibility

Extensibility and flexibility have been the focus of much of the development of this system. Different users have different needs. Trying to answer every one of those needs would be unrealistic. Instead, we provide the baseline functionality, that which is most needed, and leave open a series of extension points that application administrators can use to satisfy specific needs.

Configuring the system so it behaves slightly differently can be achieved solely by modifying configuration files, which are written in JSON. This format was chosen to avoid context switches while implementing and maintaining the system, since the results of geocoding requests, from the Google Geocoding Service, are also received in JSON format. The two files contained the default configuration of the application can be consulted in the Appendix.

For more advanced control or added functionality, Java classes must be created and added both to the class path and to the configuration files.

4.5.1 Database Configuration

Database connections are achieved through the use of JDBCs. The default database is PostgreSQL version 9.2 and the default JDBC is PostgreSQL JDBC4 version 9.2.

System administrators who need to change the database only need to include the proper JDBC for the database they chose in the classpath and change some values in the configuration file located at `config/database.js`. The fields required in the file are:

- `driverClassName` The class name for the Driver in question, which needs to be in the class path to be loaded;
- `subprotocol` The subprotocol to be used in the connection;
- `hostname` The name of the host computer holding the database;
- `port` The port the server should use to connect to the mentioned computer;
- `databaseName` The name of the database to connect to;

- `username` The user name to use in the login to the database;
- `password` The password to use in the login to the database.

At server start-up, this file will be read, and, when needed, the system will use the information read from it to connect to the database. The default configuration (without `username` or `password`) is shown in Listing 4.2.

Listing 4.2: Default Database Configuration

```
1    "driverClassName" : "org.postgresql.Driver",
2    "subprotocol" : "postgresql",
3    "hostname" : "localhost",
4    "port" : 5432,
5    "databaseName" : "novabase_geocoder",
```

At the start of development, the configuration file additionally allowed the writing of some SQL commands. These commands would be executed in response to predefined events, such as server initialization or finalization. The objective was to allow the database to have an arbitrary structure defined by the system administrator. Most commands could be defined that way, but there were some which could not.

The defined syntax would accept commands with one or more queries to run sequentially. However, some queries needed the results of previous queries to function, as was the case with the command to add coordinates to the database, which worked as follows:

1. Query the `coordinates` table for a pair of coordinates that are equal to the ones currently being added;
2. If the result set of the previous query is empty, add the coordinates, else do nothing.

This dependency of query number 2 on the results of query number 1 would require a change in the syntax of command definitions in the configuration file, introducing much more complexity than was our intention. We did not wish to insert complex logic into the JSON files, and consequently chose not to include it.

Even with this limitation, this system allowed some degree of extensibility on the database queries. However, as the project evolved, changes which further complicated this behaviour were made. One such difficulty was that the database definition changed and became more complex, which had the consequence of further complicating the SQL needed to query the database. To allow SQL into the configuration files in those new circumstances, we would need to create a Domain Specific Language (DSL), and although it would be a very simple DSL, it would not be trivial to implement. The DSL idea was discarded as being out of the scope of the project and SQL commands were removed from one of the extensibility targets, to be implemented back in further stages of the project.

Nevertheless, as long as the database structure remains as described in the earlier Sections of this Chapter, it is still possible to change databases using the configuration file.

4.5.2 Server Configuration

Extending the features of the server can be realized through the inclusion of Java classes to the class path, however, to make these inclusions known to the system, they need to be explicitly set on the configuration file located at `config/configuration.js`.

This file will be loaded at server start-up, and if it does not exist or it cannot be parsed, the server will default to a hard-coded configuration, which is exactly the same as the configuration provided by default on the file.

Information Extractors and Dispatchers These are the classes responsible for reading and writing files. An Extractor is a class that implements the `InformationExtractor` interface and a Dispatcher is a class that implements the `InformationDispatcher` interface. The default system configuration has one `InformationExtractor` and three `InformationDispatchers`:

- `CsvInformationExtractor` Enables reading from CSV files
- `CsvInformationDispatcher` Enables writing to CSV files
- `KmlInformationDispatcher` Enables writing to KML files
- `GmlInformationDispatcher` Enables writing to GML files

The syntax for declaring extractors and dispatchers is the same, and can be seen in Listing 4.3.

Listing 4.3: Configuration example for Extractors and Dispatchers

```
"extractors" : {
  "text/csv" :
    "com.novabase.geocoder.files.csv.CsvInformationExtractor",
  "application/vnd.ms-excel" :
    "com.novabase.geocoder.files.csv.CsvInformationExtractor"
},
"dispatchers" : {
  "text/csv" :
    "com.novabase.geocoder.files.csv.CsvInformationDispatcher",
  "application/vnd.ms-excel" :
    "com.novabase.geocoder.files.csv.CsvInformationDispatcher",
  "application/vnd.google-earth.kml+xml" :
    "com.novabase.geocoder.files.kml.KmlInformationDispatcher",
  "application/gml+xml" :
    "com.novabase.geocoder.files.gml.GmlInformationDispatcher"
}
```

There is a map for the extractors and another for the dispatchers. Each key in the map is a MIME-type, and each value is class path. This format limits the number of extractors and dispatchers for each MIME-type to one. The format could be changed to

allow declaring more than one by turning the values into an array of classpaths, but there is no need, since only one will be used by system.

It is, however, possible to assign the same class to more than one MIME-type, which is what happens to the types `text/css` and `application/vnd.ms-excel` (the last one being a specific case of the former). Both types are handled by the same class, and differences are supported through the use of file configurations, which will be described below.

File Configurations File configurations define how files should be read. There are two ways to specify these configurations. The first one is on the server configuration file and allows configuration on a file type basis, the second one is to define per-file configuration files and is done on a per file basis. This last one is a self contained JSON file with the same format as the object in the server configuration, but which needs to be located on the same folder as the files they specify; additionally, they need to have the exact same name (devoid extension) as the data file. For example, if we wanted to define a configuration file specific for the file `data.csv` the configuration file would need to be named `data.js` and be located on the same folder as the source file.

All file configurations should be placed in an array with the name `files`. Each position of the array will be a file configuration for some MIME-type. If more than one configuration is defined for the same MIME-type, only the last one will take effect. An example of a file configuration is in Listing 4.4.

Listing 4.4: Configuration example for File Configurations

```
"files" : [
  {
    "mimeType" : "application/vnd.ms-excel",
    "className" : "com.novabase.geocoder.files.csv.CsvConfiguration",
    "properties" : {
      "fieldDelimiter" : "",
      "fieldSeparator" : ";",
      "lineSeparator" : "\r\n",
      "address" : [1],
      "administrative_area_3" : 3,
      "country" : 4,
      "latitude" : -3,
      "longitude" : -2,
      "confidence" : -1
    }
  }
]
```

Only the first two entries of the object need to be present. The `mime-type` specifies the MIME-type to which the configuration should apply, and the `className` to which class.

The `properties` may or not be present, and it specifies entries which are specific to

each configuration. The example above defines the syntax used by CSV files written in Microsoft Excel, and the positions of geocoding information. In particular, the `address`, `administrative_area_3` and `country` properties gives instructions on how to read the full address, as was described in Section 4.1. The other three, latitude, longitude and confidence, initially were prepared to have that same functionality, but it was eventually deemed unnecessary and dropped. However, the array notation in the configuration was maintained. Negative indexes denote a position from end to start, instead of from start to end.

Depending on the context classes implementing `InformationExtractor` may chose to ignore some of the defined properties. As an example, the class responsible for reading CSV files ignores the properties `latitude`, `longitude` and `confidence` when reading user files, but acknowledges them when using files created by the system itself (which are assumed to be geocoded).

Geocoding Services New geocoding services are implemented as Java classes which extend the `GeocodingService` class. Implementations only need to write one method, `geocode`, which accepts a `GeocodingRequest` object as its argument and returns a `GeocodingResponse` object. Both classes mentioned are wrappers which give access to the underlying information unit.

Once added to the class path, they need to be declared inside geocoding strategies, which are explained below, to be used.

Geocoding Strategies Geocoding strategies define the order in which geocoding services are used, and in which cases should a response from them be accepted.

The configuration is written inside the `strategy` property, and it is composed of an array of objects, which describe geocoding services. The order by which the services appear declared in the array is the order in which the system will query them.

Each geocoding service object needs a `className` property, which value should be the fully qualified name of the class that handles the geocoding service, and three more properties which define its behaviour:

- `accept` Defines rules which instruct the strategy to accept the geocoding response of the service. If the response is accepted, geocoding services declared after the current one will not be queried;
- `cont` Defines rules which instruct the strategy to ignore the geocoding response of the service and proceed with the next service on the list;
- `fail` Defines rules which instruct the strategy to short-circuit the process and fail the geocoding of the address in the request. If the response is failed, geocoding services declared after this rule is matched will not be queried.

The rules are arrays of strings or objects. Strings are compared with the status code of the response. For instance, if a geocoding response comes with the status `SUCCESSFUL` and there is an accept rule for such a case, the process will be short-circuited and the response will be accepted.

Objects are more powerful than strings and allow control of the behaviour of the service by making use of a `rank` property. Geocoding services are not required to provide a rank with their results, in which case the rank will default to zero. In case the geocoding service does provide a rank with its results, it should be a number between zero and one, where zero would mean no confidence in the result and one would mean a perfectly matched result. One possible use of this syntax could be an `accept` rule which would only match responses with the status `SUCCESSFUL` which had a rank higher than some value, which is what happens on Listing 4.5 (`className` reduced because it was too big).

Listing 4.5: Configuration example for Geocoding Strategies

```
"strategy" : [
  {
    "className" :
      "...services.novabase.NovabaseGeocodingService",
    "accept" : [
      {
        "cases" : ["SUCCESS"],
        "rank" : 0.5
      }
    ],
    "cont" : ["NO_RESULTS", "FAILURE", "SUCCESS"],
    "fail" : []
  }, {
    "className" :
      "com.novabase.geocoder.geocoding.services.google.GoogleGeocodingService",
    "accept" : ["*"]
  }
```

The strings accepted for rules are in the following list. Additionally, the syntax accepts a string with the asterisk character, which is a wild card and an empty array which will make the rule unmatchable.

- `SUCCESSFUL` Indicates that the request was successful and there is a valid result;
- `NO_RESULTS` Indicates that the request failed because no results were found;
- `FAILURE` Indicates that the request failed to technical reasons, such as network failure.

4.5.3 Client Configuration

As this part of the project was only thought of later in the timeline, the design of its structure was not correctly defined since the beginning, and extending the client requires altering the application source files.

Nonetheless, some aspects of the Out-of-Cluster Analysis are configurable. It is possible to define the dimensions of the clusters bounding boxes, in latitude and longitude expressed in decimals, and how many locations need to be in a cluster for them to be considered valid results, either as an absolute number or as a percentage.

4.6 Server Start-up and Finalization

Start-up is what we consider to be the routines which handle server boot and initialization, until it is ready to answer requests. Tomcat already automatically loads the application, but any third-party classes which the application requires need to be loaded by the application itself.

Due to reasons explained in section 4.5, Extensibility the server needs to do some preparations before the application can be used. Some of the extensibility features can only be provided and loaded through the use of Java classes.

The start-up process will search for any classes which are specified in the configuration files and load them into memory. Objects will be created from them as necessary.

Finalization is the opposite, the routines which clean-up the server resources until everything is properly saved and closed.

When the server finishes execution, some clean up is required, namely of the JDBC Drivers. We connect to the database through provided JDBCs, which can come which built-in finalization routines. However, this is an optional feature, and some do not deregister themselves during server shutdown. To prevent memory leaks, we listen to the server shutdown event and manually deregister any Drivers still connected.

4.7 Conclusion

The implementation supports every objective initially defined, with the exception of the usability target, which was dropped as soon as we noticed the lack of resources for it.

A great part of the overhead of the implementation went to the extensibility mechanism and the reading and writing of files, which, fortunately, were the first to be implemented, which allowed us to reorganize and optimize our efforts for the final application.

Despite this, there are many aspect of the application which could be improved, and even some features that are missing, of which one is very important. These improvements and features which should be included are described in the next Chapter.

5

Conclusion

We have built an application which is capable of harnessing not only the results of various geocoding services but also those that organizations already have on their possession. The application is extensible enough to be able to fill various different needs, which increases its value, as it can be used by different organizations. While a the same time improving on the current state of the technology, by giving users the ability not only to verify but also correct the results obtained.

Mainstream geocoding services have been tested, and from those we deduced that their results do not have enough quality for heavy use by organizations when analysing bulk information. Results are often slightly displaced from their true locations, might not be obtained at all or, worse of all, be downright wrong while being accompanied by positive confidence values.

Knowing organizations have information which could improve these results, but no means to use it, we strived for the creation of an application with the intention of enabling those same organizations to make use of their information while still being able to rely on mainstream geocoding services when necessary.

The resulting application satisfies the objectives envisioned, allowing the geocoding of any number of addresses while recurring to the organization's knowledge bases and falling back to the mainstream geocoding services. However, much remains which can be improved.

5.1 Future

Although the application passes all the initially defined requirements, there is much that can be improved. These have been partitioned into two classes, features and improvements. There are some improvements which can be though of as features too, but they were moved to the improvements sections due to their lower priority in comparison with the rest.

5.1.1 Features

Client-side definition of input files For the server to be able to understand the contents of the uploaded files, it needs to have access to the structure of the files. In CSV files, for example, this means that it needs to know which columns contain which relevant information. As described earlier, these are defined in files on the server side.

This approach has the harsh limitation that, if users need to upload files with a different structure, they have to contact system administrators to change the configuration files.

Ideally, the client side of the application would have a feature where users would be able to specify this configuration, which would then be sent to the server with the file. This would allow the upload of files regardless of their structure.

Implementing this would require updating both the client side and the server. The client would have to implement the interface for the feature. The server would have to implement the saving of the configuration file with the data file. All the posterior actions would be able to proceed from there without changes.

Download files in alternative formats The current behaviour allows the download of resulting files in the same format as the source files, but sometimes it is useful to transform the source into a different file type format. For example, uploading a CSV file and downloading a GML file.

The server already supports this feature (assuming the appropriate `Extractor`s and `Dispatcher`s are available), and the client can already send the event which fires this operation, but the interface does not yet support this action, and users can not access it.

Persistent Corrections As it is now, when users correct incorrect geocoding results, those corrections are only reflected on the results file. However, it is possible to take advantage of the fact that our target base are people who know the data provided. Knowing this, and assuming that their corrections are indeed correct, it is possible to update the knowledge base with their corrections, thus increasing its size and quality.

One additional change which could be made due to this would be the creation of a new confidence value. There is already one reserved confidence value for results which come from the knowledge base, but it might prove insightful to create an extra one which

indicates that not only did it come from the knowledge base, but that it was manually inserted there by their users.

Web API Since the server already handles batch geocoding, it would be advantageous to make available a Web API. Other applications would then be able to make use of our application, just as the application makes uses of other Web APIs for its results.

5.1.2 Improvements

Improved WebSocket implementation Currently, a WebSocket connection is opened when users upload a file and is maintained through the whole session. If, for some reason, the connection breaks, the application does not reconnect and the client loses its ability to talk with the server, and vice-versa.

Though not widely tested, the server structure is prepared to support loss of connection and reconnection without disrupting any of his actions (although messages which were not sent to the client due to the broken connection are not saved for posterior update). Implementing this is a matter of the client re-attempting the connection after it is initially lost.

Database extensibility It had to be left behind because it would require too much resources to implement correctly, but assuming the resources are available it would be important for the extensibility of the whole application to go back and implement this correctly.

To properly allow arbitrary queries a small DSL would be required. The objective would be to extend the JDBC syntax to allow specific positioning of the attributes.

The JDBC syntax only allows input of variables through indexes, by placing a `?` place holder character in the query. This is not enough for our use case because the application does not know which variable to attribute to each place holder.

The idea is to expand the syntax such that it defines some names for each variable, thus allowing the DSL to use those names to specify which variable is needed at each place holder. An example would be the following query:

```
1 SELECT * FROM addresses WHERE addresses.country = ?country;
```

The `?` syntax is preserved but expanded. In this case the parser of the DSL would recognize the place holder character and read the name connected to it. This information would be fed to the application which would then know that the first place holder was to be filled with information on the variable named `country`.

The final DSL could eventually be expanded with other features, but this would be the behaviour necessary for the first intended implementation.

An added bonus of this DSL is that it would allow arbitrary database structures. Once queries become arbitrary, there would remain no need for hard-coded queries which require a pre-defined, known architecture.

Automated knowledge base updates Updates to the database have to be manually executed by systems administrators. Its maintenance could be streamlined if there was a script which could read files and update the database with its data.

The current structure of the application could be reused to implement part of this process. If the database is thought of as another kind of file, then the only requirements for the script would be an appropriate `Extractor` and a database *Dispatcher*.

Wiser selection of results from geocoding services It was mentioned in the implementation description of the algorithm for fetching results from google that, in case of multiple results, the first one is always the chosen one.

It would be possible to implement an algorithm which would analyse all the obtained results and choose the more appropriate one. Whether this is useful and reliable enough is something that would be needed to be researched first.

Automatic calculation of kilometres to decimal latitude/longitude As mentioned before, when describing how to change the size of the bounding boxes for the Out-of-Cluster Analysis, distances have to be specified as latitude and longitude in decimal format. This is because in different locations, one unit of latitude or longitude corresponds to different distances. However, this change is stable for every degree, so it is possible to design an algorithm that, knowing a location, would transform distances in kilometres to their respective latitude and longitude quantities.

This would, however, require some way of knowing the ratio from latitude/longitude to kilometres of the specific locations.

Client configurations on separate files In its current state, the source files of the client application need to be changed directly to allow for customization. The ideal way of doing this would be to have separate Javascript files which contained the necessary variables (for configuration specific aspects of analysis) and functions (for custom analysis), which would then be included in the resources of the page, allowing our main script to load its configuration from there, to the client.

Bibliography

[Bat13] Batch geocode, january 2013. Visited on 27 August 2013.

[Bin13] Bing maps rest services, january 2013. Visited on 27 August 2013.

[Bul13] Bulk geocoder, january 2013. Visited on 27 August 2013.

[CC02] Tim Churches and Peter Christen. Preparation of name and address data for record linkage using hidden Markov models. *BMC Medical Informatics and Decision Making*, 16:1–16, 2002.

[CCW04] Peter Christen, Tim Churches, and Alan Willmore. A probabilistic geocoding system based on a national address file. In *Proceedings of the 3rd Australasian Data Mining Conference, Cairns*, 2004.

[CCZ02] Peter Christen, Tim Churches, and Justin Xi Zhu. Probabilistic name and address cleaning and standardisation. Citeseer, 2002.

[CSV13] Common format and mime type for comma-separated values (csv) files, september 2013. Visited on 23 March 2013.

[CT03] Michael Cayo and Thomas Talbot. Positional error in automated geocoding of residential addresses. *International journal of health geographics*, 2(1):10, 2003.

[Fin13] Find latitude and longitude, january 2013. Visited on 27 August 2013.

[Gat89] Anthony C Gatrell. On the spatial representation and accuracy of address-based data in the united kingdom. *International Journal of Geographical Information System*, 3(4):335–348, 1989.

[Goo13] Google maps api web services, january 2013. Visited on 27 August 2013.

[GPS13] Gps visualizer, january 2013. Visited on 27 August 2013.

[GWK07] Daniel W Goldberg, John P Wilson, and Craig A Knoblock. From text to geographic coordinates: the current state of geocoding. *URISA journal*, 19(1):33–46, 2007.

[Rat01] Jerry H Ratcliffe. On the accuracy of tiger-type geocoded address data in relation to cadastral and census areal units. *International Journal of Geographical Information Science*, 15(5), 2001.

[RDA01] Marie Christine Roy, Olivier Dewit, and Benoit A Aubert. The impact of interface usability on trust in web retailers. *Internet Research*, 11(5), 2001.

[Top13] Topoly, january 2013. Visited on 27 August 2013.

[VMN01] Luís Veigas, Miguel Marques, and Jorge Neves. Georeferênciação Automática De Endereços Portugueses - GAP. page 8, 2001.

Listing 1: Configuration file for the application

```
{
  "extractors" : {
    "text/csv" :
      "com.novabase.geocoder.files.csv.CsvInformationExtractor",
    "application/vnd.ms-excel" :
      "com.novabase.geocoder.files.csv.CsvInformationExtractor"
  },
  "dispatchers" : {
    "text/csv" :
      "com.novabase.geocoder.files.csv.CsvInformationDispatcher",
    "application/vnd.ms-excel" :
      "com.novabase.geocoder.files.csv.CsvInformationDispatcher",
    "application/vnd.google-earth.kml+xml" :
      "com.novabase.geocoder.files.kml.KmlInformationDispatcher",
    "application/gml+xml" :
      "com.novabase.geocoder.files.gml.GmlInformationDispatcher"
  }
  "strategy" : [
    {
      "className" :
        "com.novabase.geocoder.geocoding.
          services.novabase.NovabaseGeocodingService",
      "accept" : ["SUCCESS"],
      "cont" : ["NO_RESULTS", "FAILURE"],
      "fail" : [],
      "normalizer" : {
        "className" :
          "com.novabase.geocoder.geocoding.normalization.SimpleNormalizer",
        "properties" : {
          "stopWords" : "config/stopwords.js",
          "replacementChars" : "config/replacement_characters.js",
          "replacementWords" : "config/replacement_words.js"
```

```
        }
      }
    }, {
      "className" :
        "com.novabase.geocoder.geocoding.
          services.google.GoogleGeocodingService",
      "accept" : ["*"]
    }
  ],
  "files" : [
    {
      "mimeType" : "application/vnd.ms-excel",
      "className" : "com.novabase.geocoder.files.csv.CsvConfiguration",
      "properties" : {
        "fieldDelimiter" : "",
        "fieldSeparator" : ";",
        "lineSeparator" : "\\r\\n",
        "address" : [0],
        "administrative_area_3" : 1,
        "country" : 2,
        "latitude" : -3,
        "longitude" : -2,
        "confidence" : -1
      }
    }, {
      "mimeType" : "application/vnd.oasis.opendocument.spreadsheet",
      "className" : "com.novabase.geocoder.files.csv.CsvConfiguration",
      "properties" : {
        "fieldDelimiter" : "\\\"",
        "fieldSeparator" : ",",
        "lineSeparator" : "\\r\\n",
        "address" : [0],
        "latitude" : -3,
        "longitude" : -2,
        "confidence" : -1
      }
    }
  ]
}
```

Listing 2: Configuration file for the database

```
{
    "driverClassName" : "org.postgresql.Driver",
    "subprotocol" : "postgresql",
    "hostname" : "localhost",
    "port" : 5432,
    "databaseName" : "novabase_geocoder",
    "username" : "postgres",
    "password"  : "pass",
    "instructions" : {
```

```
10          "query" :
11        "SELECT
12          o.address, o.locality, o.administrative_area_1,
13          o.administrative_area_2, o.administrative_area_3, o.country,
14          c.latitude, c.longitude
15        FROM
16          addresses_components_original as o INNER JOIN
17          addresses_components_norm as n USING(coord_id) INNER JOIN
18          coordinates as c ON(n.coord_id = c.coord_id)
19        WHERE
20          (n.address = ? OR (n.address IS NULL AND ? IS NULL)) AND
21          (n.locality = ? OR (n.locality IS NULL AND ? IS NULL)) AND
22          (n.administrative_area_1 = ? OR
23            (n.administrative_area_1 IS NULL AND ? IS NULL)) AND
24          (n.administrative_area_2 = ? OR
25            (n.administrative_area_2 IS NULL AND ? IS NULL)) AND
26          (n.administrative_area_3 = ? OR
27            (n.administrative_area_3 IS NULL AND ? IS NULL)) AND
28          (n.country = ? OR (n.country IS NULL AND ? IS NULL));"
29      }
30  }
```