Institut für Verteilte und Parallele Systeme
Abteilung Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Studienarbeit Nr. 2354

Probabilistic map representation using GeoTools

Daniel del Hoyo

Course of Study:	Computer Science
Examiner:	Prof. D. Kurt Rothermel
Supervisor:	M.Sc. Pavel Skvortsov
Commenced:	24. October 2011
Completed:	29. March 2012
CR-Classification:	C.2.4, E.2.0, H.3.4, I.3.5

Abstract

With the current raise of many different smartphones devices and tablets, which have among other functions, a GPS sensor, the ability to make use of services based in the user's location is increasing, as the number of applications offering this type of service. In addition, the number of social networks which allow users to share their location is also increasing.
To avoid any kind of leak that could lead to violate user's privacy or security, location privacy techniques should be implemented. Obfuscation will be used in order to avoid the usage of third party solutions. The generation of cloaking regions allows users to define an associated privacy police and configuration, associated with every different type of feature, when map-aware knowledge is available.

This paper presents different methods of probabilistic map representations and an obfuscation technique used when unveiling user's position.

Keywords: GeoTools, Cloaking, Obfuscation, Probabilistic representation, Map-awareness, Location privacy.

Contents

List of Figures

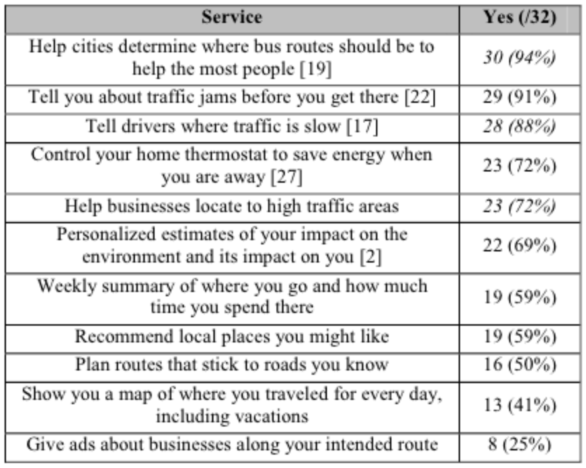
List of Tables

List of Equations

1 Introduction

1.1 Motivation

A huge number of new applications that use location features are being developed constantly, and they are now widely spread and available on the market. This software permits to the user, among other, realize queries about where is the closest restaurants that better fit her needs, or knowing if her friends are somewhere close to her actual position.



Service	Yes (/32)
Help cities determine where bus routes should be to help the most people [19]	*30 (94%)*
Tell you about traffic jams before you get there [22]	29 (91%)
Tell drivers where traffic is slow [17]	*28 (88%)*
Control your home thermostat to save energy when you are away [27]	23 (72%)
Help businesses locate to high traffic areas	*23 (72%)*
Personalized estimates of your impact on the environment and its impact on you [2]	22 (69%)
Weekly summary of where you go and how much time you spend there	19 (59%)
Recommend local places you might like	19 (59%)
Plan routes that stick to roads you know	16 (50%)
Show you a map of where you traveled for every day, including vacations	13 (41%)
Give ads about businesses along your intended route	8 (25%)

Figure 1: Willing to reveal position according to the service [26]

Unfortunately, this new services come associated to a privacy problem referring to the location of the user. Letting anyone know where you are in every moment leads to the lost of privacy at all, and even drives a malicious users to be able to make a physical attack. Also, users are nowadays more concerned in every subject referring their privacy, as Figure 1 and some studies shows [34].

1.2 Objectives

The main topic of this study thesis is to work with the different features of the objects, and to implement an obfuscation method to assure location privacy. The next associated task and topics will be covered:

- Overview the GeoTools library.
- Study of different location privacy methods (like the ones shown in Figure 2, and obfuscation techniques

- Obtaining the different features of objects, in order to calculate the intersection between regions
- Representation of Features in a graphic map, using GeoTools
- Calculate and represent cloaking regions when using obfuscation techniques
- Implement different probabilistic map representation techniques
- Calculate and graphically represent probabilistic location values

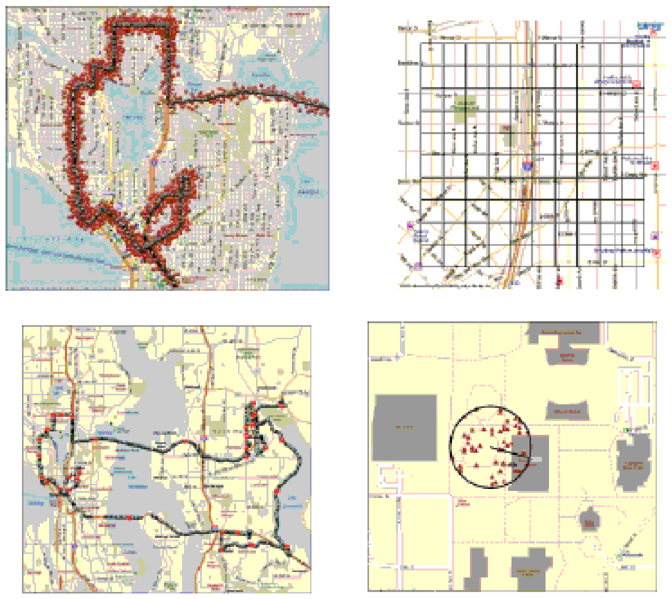


Figure 2: Different location privacy techniques [26]

1.3 Scenario

The algorithm and software developed were realized in order to guarantee the user's location privacy. To do so, imprecise user's position is represented as a circular shape. It's important to know how secure it is, i.e. which map objects lay under it (are intersected). Once we know this it would be possible to decide how the privacy radius should be increased, assuring the privacy of the user.

A typical usage scenario could be a mobile client, with some privacy settings configured, and who's walking inside a random city. When the user tries to use a location-based service, the solution we present here will be applied at the user's side, revealing in a secure way the geolocation of the user, according to her privacy settings, when possible.

1.4 Structure

This thesis study will be structured as follows:

- Study of different tools and data structures that are at our disposal.
- Analysis of the existing related work, that concerns the goals of our study.
- Formulation and development of a solution to this study thesis topic.
- Analysis and evaluation of the proposed solution.
- Conclusions and possible future work.

2 Background

As first point of this work,two of the most common used toolkits(GeoTools and OpenLayers) will be analyzed, and also some of the most common an extended formats, in order to determine which one of them suits better to our objectives.

2.1 GeoTools

GeoTools [1] is a Java toolkit oriented to the development of GIS software. It has different methods to manage geographic information, and also to draw geometries.
GeoTools offers support for a wide number of formats, including raster formats, vector formats (shapefile), xml bindings (gml,kml), and database support. The data structures used in this library are based on Open Geospatial Consortium specifications shown in figure 3.

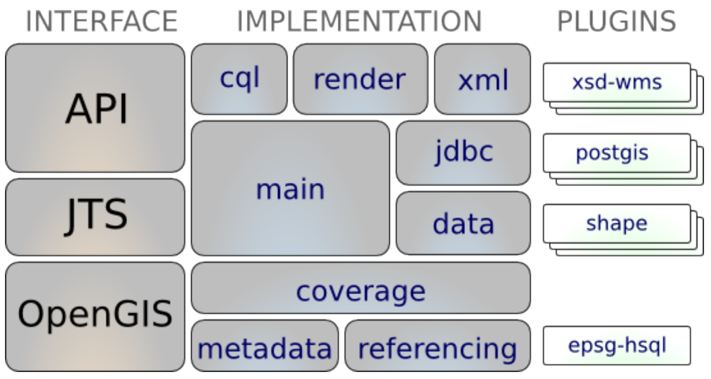


Figure 3: OGC specifications [1]

GeoTools core features are the following [1]:

- Definition of interfaces for key spatial concepts and data structures.
- A clean data access API supporting feature access, transaction support and locking between threads.
- A stateless, low memory renderer, particularly useful in server-side environments.
- Powerful schema-assisted parsing technology using XML Schema to bind to GML content.
- Open plug-in system.
- Extensions that can provide additional capabilities built using the spatial facilities of the core library.

2.2 OpenLayers

OpenLayers is an open source JavaScript library. It provides support for displaying map data in web browsers, in a similar way like Microsoft or Google do it with Bing Maps or Google Maps respectively. As one of the Open Source Geospatial Foundation tools, it's able to communicate with many different protocols.

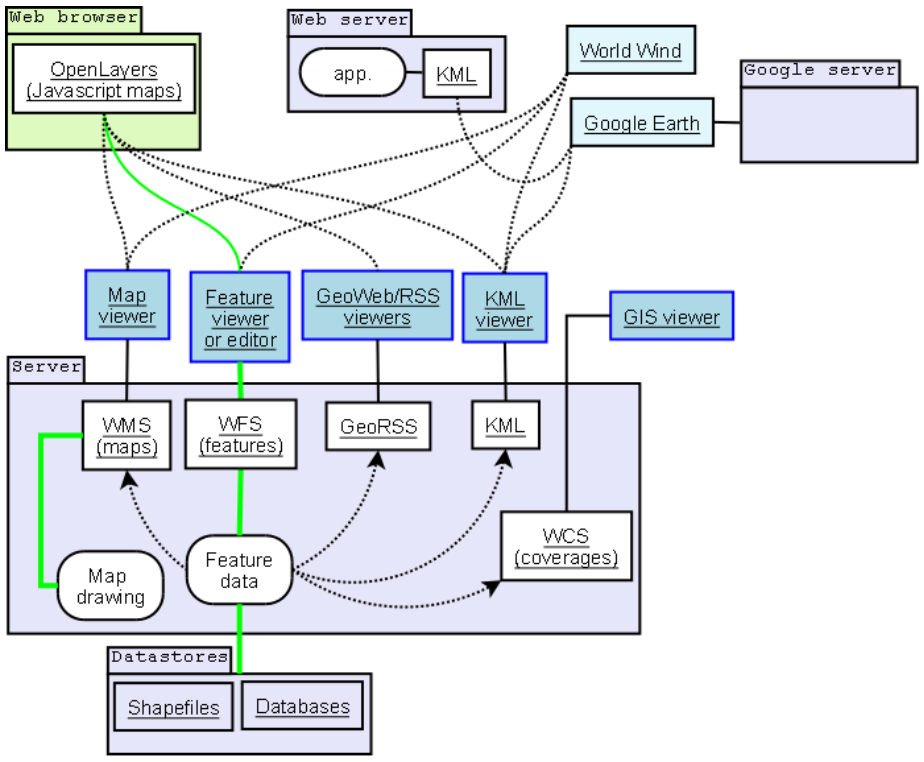


Figure 4: Different communication between protocols with OpenLayers [32]

As shown in figure 4, it provides the following features:

- Load data from many different sources. Some of them are:
  - Web Map Service
  - Web Feature Service
  - Google Maps
  - OpenStreet Map
  - ArcGIS Server
  - Yahoo! Maps

- Provides support for different data formats:
  - GML
  - KML
  - GeoJSON
  - GeoRSS

Many projects, like OpenStreetMap, use it for different task. As we're not interested in displaying info into web browsers, a deeper analysis of its characteristics won't be realized.

2.3 Data formats

With the objective to select the most suitable data format to our purposes, an study of the different available data formats to represent information about maps has been done.
It has been analyzed the ability and precision representing geographic limits, geometric limits and properties, and some other interesting characteristics, like the ability to extend the format in order to add our own properties.

The study done is explained in the following sections:

2.3.1 GML, KML and OpenStreetMap formats

Geography Markup Language (GML)

GML is an XML grammar defined by the Open Geospatial Consortium in the year 2000 in order to express geographical features [7]. It's an open, vendor-independent standard , used as a modelling language for geographic system, as well as an open interchange format for geographic transactions. As it's XML-based, its format is human readable, which gives an advantage when analyzing and understanding it.
It offers different primitives, which include features, geometry, coordinate systems and topologies among others.

For GML, Feature and geometric objects are different. A feature is a physical object, and it can be defined(but it doesn't have to) by one or more geometric objects. The most important ones are *Point, LineString* and *Polygon*, but more complicated ones, like *Curves* can be defined.

A *Point* is defined by a tuple of an x and y coordinate. A *LineString* is defined by two or more points, with linear interpolation between them. A *Polygon* is defined by an inner and outer rings. A *Ring* can be defined as a *LineString*, where the beginning and the ending of it is the same. Outer rings express the outer part of the polygon, inner rings refer to the inside part of it.

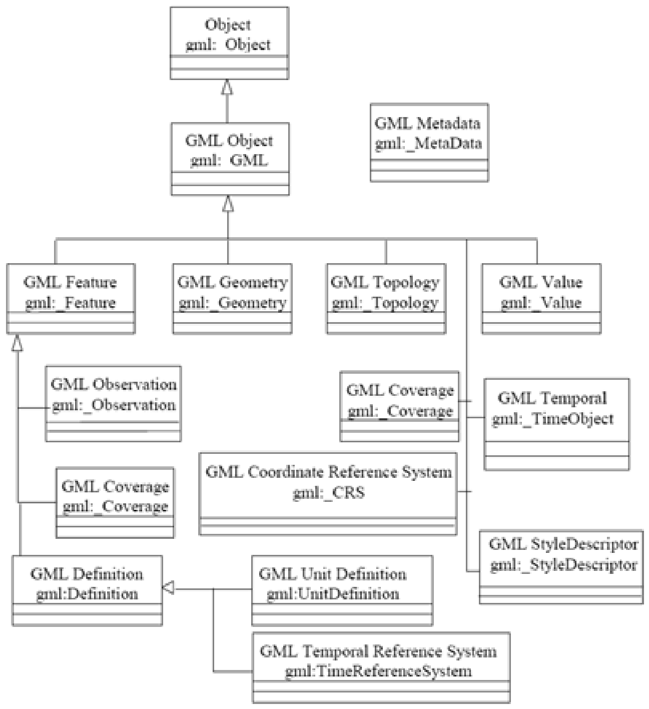


Figure 5: GML hierarchy (UML diagram) [7]

In order to refer just to the object whose data the user is interested in (like hotels and restaurants for a tourist software, for example), application schemas can be defined, based on the domain of interest.
As ISO standard (ISO 19136:2007), it's widely used nowadays. Developers and researchers (OpenGis, Galdos Inc., Ionic Software, Cubex...), Governmental agencies(US census bureau, Ordnance Survey UK, National Resources Canada) and geographic software developers (Esri, MapInfo, Oracle, NTT Data) base their work on GML.

GML strength resides in its XML-based format. As XML, the data integrity can be easily checked, easy read and edited(just a text editor is needed), and as XML is very extended, is easy to work with (many APIs and classes available) and to integrate with non-spatial data.

Once the GML data is obtained, a transformation between the data and some vectorial graphic format (like SVG), is needed. Many software offer this functionality, like GMLDataStore combined with GeoTools, for example.
Some examples of very extended software or related formats based on GML are:

Web Features Service (WFS)

The Open Geospatial Consortium provides an interface standard (WFS) [8], that allows requests for geographical features. GML passes data back and forth between the WFS and the client. Static and dynamic interfaces are available for this purpose.

Google Maps: GeoXml

As Google Maps is very used and extended, many extensions have been developed in order to work with GML spatial data. One of them is GeoXml [9], which provides client-side parsing of GML, including GML from WFS servers, and some other options to work with.

GeoRSS

In order to encode location as part of a web feed, GeoRSS was raised as emerging standard. Location consist of geographical points, lines, and polygons of interest and related feature descriptions.

CityGML

CityGML [10] has been developed as an information model for the representation of sets of 3D urban objects. Actually, an OGC candidate specification has been released for public comments (August 2011).

Keyhole Markup Language (KML)

KML is also an XML notation for expressing geographical annotations and visualizations both in two and three-dimensional maps. The Open Geospatial Consortium adopted it as a standard in 2004.
Originally, it was developed for using with Google Earth by Keyhole Inc, which was acquired by the Google company. Today, this language is used for the visualization of geographic information tailored by *Google Earth*, although some open source Earth programs, like *Marble*, are starting to use it. It can be used to carry GML content.

It uses a tag-based structure with nested elements and attributes. Many free tools and documentation are available online, including KML references and tutorials on Google Code. The KML files can be compressed using a zip utility, giving us the KMZ files

As not all the structures contained in GML can be transformed to KML, and in this work we are not interested in developing an Earth browser, KML won't be studied in detail.

OpenStreetMap (OSM)

As studied in the previous Studienarbeit [12, p15], the OpenStreetMap-format (OSM from now on) is a collaborative project [13] to create free and editable maps.

OSM doesn't imply just a data format, but different ones and databases. They can be consulted through the OSM API, which offers a REST interface, which can be used to obtain the data we're interested in, or the different tiles, specifying the longitude, latitude, and the zoom level after a previous conversion.

Some possible geometric objects in OSM could be *Nodes*, *Ways*, *ClosedWays*, *Relations*, and *Tags*. There's an analogy between *Node* and *Point* in GML, *Way* and *LineString* and also between *ClosedWay* and *Polygon*. *Relations* are aggregations of different objects, and *tags* can be used to store meta-data.

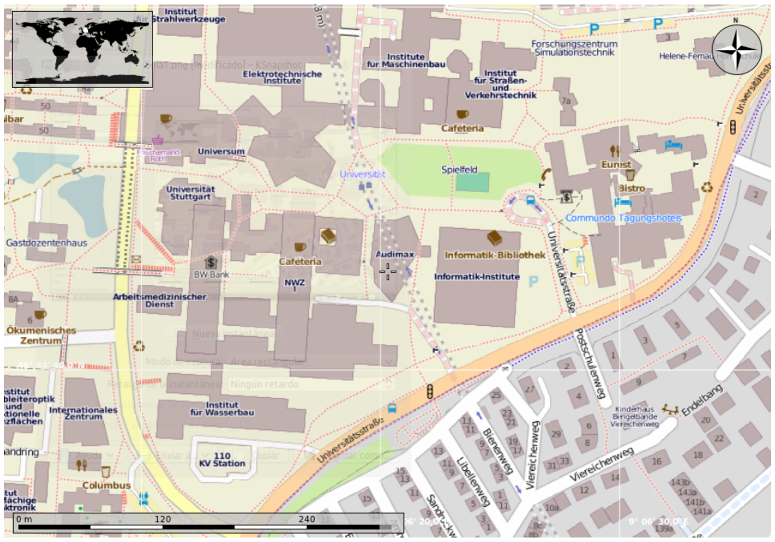


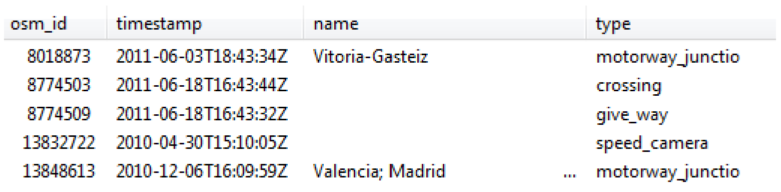
Figure 6: OpenStreetMap used in the free software 'Marble'

2.3.2 Shapefile format

The ESRI Shapefile format, SHP, is a proprietary data format [11] for geographic information systems. Nowadays, there's plenty of geographical information available in this format. Due to the fact that it's well documented, it's a de facto standard. It's used to describe geometries, like points and polygons. Each item may also have attributes.

A shapefile is a set of several files. These are:

- **.shp**: the geometry itself.
- **.shx**: a positional index of the feature geometry, to allow seeking.
- **.dbf**: columnar attributes for each shape.



osm_id	timestamp	name	type
8018873	2011-06-03T18:43:34Z	Vitoria-Gasteiz	motorway_junctio
8774503	2011-06-18T16:43:44Z		crossing
8774509	2011-06-18T16:43:32Z		give_way
13832722	2010-04-30T15:10:05Z		speed_camera
13848613	2010-12-06T16:09:59Z	Valencia; Madrid ...	motorway_junctio

Figure 7: Attribute examples in a .dbf file

Also, some optional files can be defined, in order to specify the coordinate system, character encoding, or metadata in XML format.

SHP data is generated by the U.S. Census Bureau, therefore, free maps of US can be obtained. Unfortunately, it's not so widely extended in the rest of the world, but also maps from any other place can be obtained (previous payment) from some entities like geofabrik.de.

There are different shape types that can be used. Some of the most important ones are:

- **Point**: defines a point, using an X and Y coordinate.
- **MultiPoint**: represents a set of points.
- **PolyLine**: an ordered set of vertexes that consist of one or more parts. A part is a connected sequence of two or more points, but they don't have to be connected one another.
- **Polygon**: one or more rings. A ring is a connected sequence of four or more points that form a closed, non-self-intersecting loop. Inner and outer ring can be defined with this shape.
- **MultiPatch**: consist of a number of surfaces patches. Each surface patch describes a surface, which can be rings, triangle stripes, or triangle fans.

Measures can be added, obtaining the *PointM* and *MultiPointM* types among others.

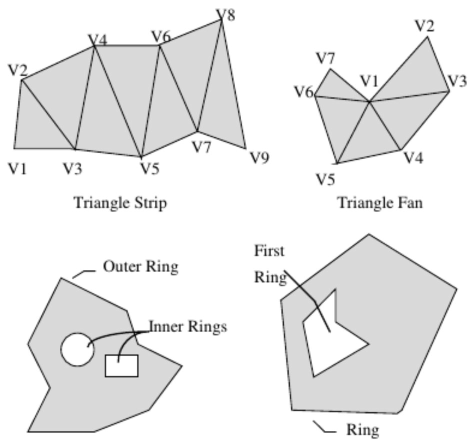


Figure 8: Example of *MultiPatch* parts [11, p21]

Shapefile has some limitations. The edges of a *PolyLine* or a *Polygon* are defined using points. Therefore, the spacing between them determines the scale for which the data are useful. If smooth shapes want to be achieved, additional points would be required. Also, this format lacks the capacity to store topological information or any feature topology.
The data type field with is also highly correlated with the geometric boundaries of features. Maximum length of floats and doubles is 13 digits, but usually the free available maps make use only of 6 or 7 digits.
Circular arc curves are not supported on shapefiles This doesn't make the shp format the best suitable when we are interested in geometric limits.

It is used in many software, such as gvSIG, Kosmo, Shape Viewer, uDig or GRASS GIS.

2.3.3 Other formats

Historically speaking, the hardware development supposed a big impulse to the software development of generic cartographic applications. Therefore, from the early 60s, many data formats have been developed since then, and they served as proof of concept and inspiration of the data formats used nowadays.
Many countries and state agencies wanted to be on the shape edge of the geospatial revolution, investing resources on developing new data formats.

Some of the most famous(but almost no used any more) are:

Spatial Archive and Interchange Format (SAIF)

The SAIF format [2] has been considered one of the precursors of the GML standard. Nowadays, it only has an historic interest. It was defined in the 90s as a self-describing extensible format to support the storage of geospatial data. Formed by two major classes, the Class Syntax Notation (a data definition language), and the Object Syntax Notion (used to represent the object data), it became a Canadian National Standard in 1993. It was taken into account on the initial version of the Open Geospatial Interoperability Specification, what became later the Open Geospatial Consortium (OGC).

Canadian Council on Geomatics Interchange Format (CCOGIF)

The Canadian Council on Geomatics Interchange defined the CCOGIF format [3], a very general medium in which to represent a data model. It's focused on a very low level, therefore, a more sophisticated mapping file is required in order to make full use of the data.

Digital line graph (DLG)

A digital line graph [5] is a cartographic map feature (digital vector) distributed by the US Geological Survey. They are distributed in three scales: large, intermediate, and small, containing different categories of features depending on the scale. These can be: Public land survey system, boundaries, transportation, hydrography, hypsography, non-vegetative features, survey control and makers, man-made feature end vegetative surface cover. Usually, they come in SDTS format.

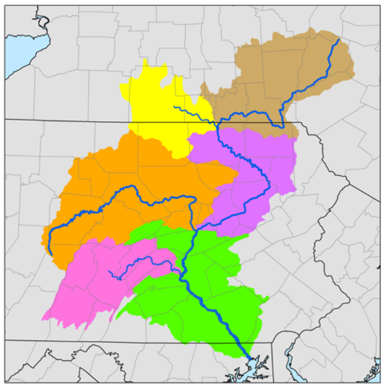


Figure 9: Susquehanna River watershed sub-basins produced with DLG data [35]

On the next following pages, more up-to-date formats will be analyzed.

Simple feature access (SFA)

It's an OpenGIS and ISO standard (ISO 19125), that specifies a common storage model of geographical data using text or binary. The geometries are also associated with a spatial reference system, attributes, methods and assertions.

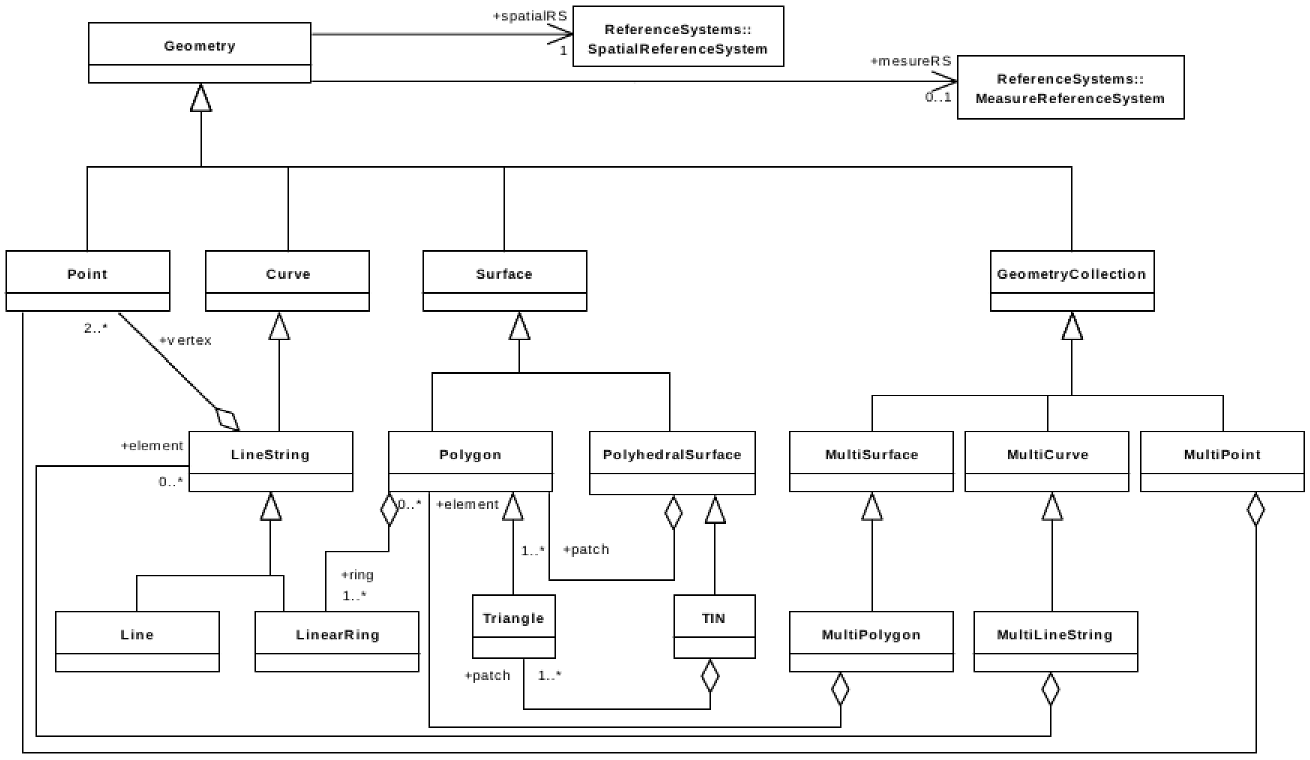


Figure 10: Geometry class hierarchy [6]

Part of the most important geometries that can be defined are:

- **Point**: defines a point, using an X and Y coordinate.
- **Curve**: sequence of points, with the subtype of Curve specifying the form of the interpolation between them.
- **LineString**: A curve with linear interpolation between points

- **Surface**: Single patch that is associated with one exterior boundary and 0 or more interior boundaries.

- **Polygon**: Planar surface.

Also, other objects are available (MultiSurface, MultiPoint).

The representation of the objects can be done using well-known-text (a text-markup-language notation), or using a binary format. The text representation of geometry are defined using the BNF notation. The binary representation permits geometric objects to be exchanged between an SQL client and an SQL-implementation in binary format.

GeoTools offers support for Simple Feature. The GML simple feature standard is close to this data format.

GPS track database

In the last years, the amount of devices that can connect to a GPS station, have increased and spread all over the world. Due to this, GPS information can be reused to make queries about geographic information.
Using an XML schema, for example, GPS information can be described. The GPX open format [14] allows as to define waypoints, tracks, and routes. Tags storing location, time and so on can be defined, making this format really useful to interchange data between GPS devices and different software applications.

A geodatabase is a spatial database designed to store, query and manipulate geographic information of low dimensionality. Basically, it's a type of a spatial database, with optimizations for 2 and 3 dimensions, and Euclidean distance. It can handle point, line, or polygon data types. The reference system can be specified.
Different geospatial data types can be specified, not just points. For example, using SQL syntax, a polygon can be added to a table using the following command:

```
INSERT INTO Districts (DistrictName, DistrictGeo)
VALUES ('Stadtmitte', geometry::STGeomFromText
('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));
```

Although many geodatabases can be queried using SQL, the spatial data only be accessed using specialized software. But, as a spatial database, it gives us all the advantages of a relational database management system. Many commercial and open source DBS offer support to store geospatial data, like PostgreSQL, MySQL, and Oracle.

The main difference between this format and the other ones seen before is its

capability to store feature topology.
It should be noticed that the user will query for points, and geometries. Therefore, the higher the number of registers and points our database has, the better the geographic limits and the results we will have.

GeoJSON

It's an open format for encoding geographic data structures. Its name is related to the fact that it's based on JSON. The advantage between JSON and XML is that the JSON format is more compact. The data types supported by GeoJSON [15] are:

- Positions
- Point
- MultiPoint
- LineString
- MultiLineString
- Polygon
- MultyPoligon
- Features
- Geometry Collections

Also, the geographic coordinate system can be specified.

It's an extended format. Some well-known services, like OpenLayer, or the geolocation capability from Twitter, are served using GeoJSON.
Apart from the data formats seen before, there are also other file formats designed to work with geospatial data. Despite most of them are not open-source, and they belong to a governmental organization, some of the most common are the following ones.

National Transfer Format (NTF)

It was designed in 1988 by the British Standard Institution. It is now the standard transfer format for the Ordnance Survey digital data.

Systematic Organization of Spatial Information (SOSI)

Samordnet Oppleg for Stedfestet Informatsjon (SOSI), known in English as Systematic Organization of Spatial Information includes standardized definitions for geometry and topology, data quality, coordinate systems, attributes and metadata. Developed by the Norwegian Mapping and Cadastre Authority, is used for exchange of geographical information in Norway.

2.4 Analysis

As we're going to work with the GeoTools toolkit, the data formats we're going to use should be compatible with it. The SimpleFeautres GML and the Shapefile format offer a well-documented interface and a large number of compatible classes.
Another factor that should be considered is the available data sources, documentation, and if it's widely spread and used in real software and hardware. Both GML, Shapefile, and OSM are well documented and widely used. Also, KML is used by the Google Earth software. There are many shapefiles available to work with. The documentation and use of a GPS track database basically depends on de database management system we're working with.

Some of the most important files will be analyzed. The rest of them have been discarded, due to the fact that they're a proprietary format, or just because they don't suit our needs.

The object types that a format offers will determine the way we work with it. Most of them can include metadata information. The different objects offered, broadly speaking, are as follows:

Object	**SimpleFeatures**	**SHP**	**OSM**	**GPS track DB**
Basic	Point LineString Polygon	Point PolyLine Polygon	Nodes Ways ClosedWay	Point Line Polygon
Complex	Yes	Yes	-	-
Metadata	Properties	Attribute	Tags	-

Table 2.1: Supported types of different formats

To summarize all this, a comparative table with the different available formats has been developed:

Filetype / Property	GML	KML	OSM	SHP	GPS track DB
GeoTools compatible	✓	✓	✓	✓	✓
3D	✗	✓	✗[1]	✓[2]	✓
Geographic limits	✓	✓	✓	✗[3]	✓[4]
Feature topology	✗	✗	✗	✗	✓
Graphic details	medium	medium	high	high	low
Data format	XML	XML	XML/binary	binary	binary(queries)
Metadata	✓	✓	✓[5]	tickYes	✗[6]
Availability	high	medium	high	low[7]	high[8]

Table 2.2: Comparison of different formats

Every format summarized in this table are consistent with the GeoTools toolkit we are using. Therefore, they should be considered as objects of study. Furthermore, as already mentioned, all of them are open-source formats which make them possible candidates for the study thesis we are conducting.

On the first place, we can notice that several of them have support for 3D representation. Only GML and OSM do not support it (although there are available extensions to do so). Since we are only interested in representing 2-dimensional maps, in order to study and normalize probabilities, a priori it's not factor that is relevant to this work, although it might be interesting in a future.

Some other factors are very relevant to our study. The first one is the geographical limit. Our program will work with the location of different users. The more accurate and reliability we are able to obtain from our data structure, the better and more accurate the results and calculations that the software offers will be. It is necessary to be able to determine exactly if a user is inside the building, in the limits of it, or just in the vicinity.

[1]OSM-3D [37] can be used for this purpose. This project is being developed by the university of Heidelberg.[38]

[2]Using software and format extensions, like 3D Analyst [36]

[3]Fails when scaling, as explained in section 2.3.2

[4]Amount of points depending, see section 2.3.3)

[5]Tags can be used for this. (section 2.3.1)

[6]DB could be extended, depending on the DBMS. See section 2.3.3

[7]Maps can be acquired from geofabrik.de

[8]Maps can be converted from OSM data. Many open source software, like RouteConverter [39] is available to do so

In addition, we will work with different scales, therefore, the data and service quality should not be decrease when reducing the scale and increasing the zoom level. We're more interested in geographic limits of the object rather Than its geometric information, in order to work with probabilities (i.e., knowing if a point is inside a building or not).
The geographic information that our data structure is able to offer is not a relevant factor in our case, but it also should be taken into account.
Also the ability to introduce and work with metadata in the different formats we are studying.

Another important factor, and necessary to be considered, is the availability and the up-to-date data we have. We are interested in our program to work with updated maps and data, easy to get and download, either because their availability in the format we're using, either by their free or low price. In this aspect, OSM format is the one who better suit our needs, because many maps are accessible and downloadable for free, and are being permanently updated by its users.

The data format itself (binary, text...) doesn't suppose any problem, since our toolkit is capable of handling a wide range of them. However, it is true that the more alternatives it offers, the more methods to process the data we will have.

Therefore, we are interested in maximizing the accuracy of geographic boundaries, and the availability of the data, as main factors, and in these two should reside the strength of the data format we have chosen.
Once these requirements have been fulfilled, other characteristics such as geographic information, or the format it uses, can be taken into account.

With all these we can conclude that the format that best suits our needs is OSM, as it can be checked reviewing the realized analysis summarized in the present table.
However, the previous study thesis source code we have, works with the shapefile format. Due to the fact that these format also suit to our needs, and such a change in the implementation is not strictly necessary, and will derive in an extra investment of effort and time, we will keep using this data format.
It would be interesting to support also the OpenStreetMap format. When developing for shapefiles, and using the GeoTools toolkit, is easy to convert OSM to SHP, and make use of them.

With the purpose of possible future developments and improvements, the feasibility of adapting the structure of our program to the Openstreetmap data format will be analyzed later on.

2.5 Basics of location privacy

As it has already been mentioned, the upcoming development making use of the geographical position of the user, and also the development of new, diverse, and useful services, has brought with it some new problems, relating location privacy issues among others.
Location privacy can be defined as the ability to prevent other parties from learning one's current or past location [17]. Therefore, any system that obtains or works with location data is susceptible to have some location privacy flaws.

Many different methods have been developed in order to assure location privacy. Basically, they can be classified in 4 different main categories:

- **Regulatory strategies:** governmental agencies stablish with diverse laws how and under what conditions the location data can be used. Due to the fact that this measures are out of the developer control, and it would be possible to any coder(either by a misuse of the software, either because regulatory strategies are different in his own country) to not comply with the regulatory requirements, they won't be analyzed in detail in this paper.

- **Privacy polices:** a set of rules are legislated which define how information is to be stored and used. Also, it can be established when can the data be revealed. As the location-based service (LBS) is not obliged to obey those rules, it's not applicable in our actual scenario where we're working at.

- **Anonymization methods:** The idea behind is to hide the relationship between the user's identity and its location. Some strategies, like k-anonymization, fall under this profile, and would be studied in the following lines.

- **Focus on the location data itself:** Degrade the quality of the location information. Doing this, the user doesn't inform of a concrete position, but informs of a region. The obfuscation techniques will fall into this category.

Anonymization smethods:

One of the most used anonymization strategies is k-anonymity [29]. This technique is based in restricting location updates only when the user is inside a sensitive area. The position would only be send if inside that zone, are at least $k-1$ with the same characteristics. Of course, if it's not a very dense zone, or a widely used service, the region would be bigger, resulting in a bigger vagueness of the location and a degradation of the service.

Sadly, nowadays LBS are not spread enough, so it's very likely that we would have to work with broad regions.

Most published k-anonymity approaches use a trusted third party as an anonymizer where the implementation could be based on a centralized [3] or distributed architecture. One of the important challenges in k-anonymity is to find k-1 other users to keep the anonymity. To calculate the different k-zones, a sensitive map has to be available,with at least k-sensitive areas.

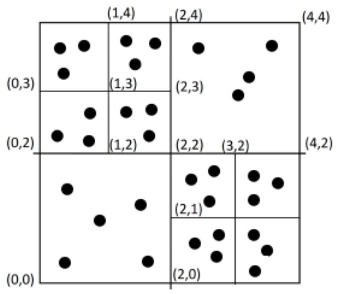


Figure 11: Map subdivision for k-areas of k=3 [46]

Despite of the fact that different alternatives solutions based on this idea has been purposed (working in non-trusted systems [18], or even making use of collaborative k-anonymization [19], either using a third-party or peer-to-peer approaches), none of them can be applied in our defined scenario, for the reasons already told.

Also, there are some inherited additional problems, like the computational cost of this solution, or the need to take into account the semantic in order to avoid the location of an user inside a k-zone. Some other problems with k-anonymity approaches are the reduction of accuracy and the need for a trusted third party.

Cryptographic approaches:

In protecting the data, we can make use of cryptographic and ciphering tools [20] [21]. This protocols provide us some classic services, like authentication, secrecy, and nor-repudiation. One of the most common methods implies the use of session tickets, as the 2PLoc [22] solution does.

Also, private information retrieval techniques (PIR), can be used. This guaranties that the location server will answer our queries without knowing what kind of information are we asking for.
Unfortunately, all this increase not only the computational traffic, but also the data traffic, producing bottle necks in the location service. As we are going to use our software in mobile devices and smart phones, we want to keep those to factors as low as possible, so this kind of solutions won't be considered.

Focusing on the data itself:

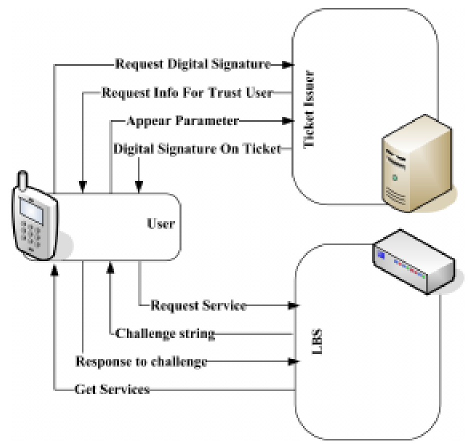


Figure 12: Cryptographic scheme of the 2PLoc protocol [22]

The dummies generation strategies are based in creating some dummy locations, and then send them beside the real location to the server provider. There are many different and a huge variety of techniques, between the simple ones, who only generate random dummy tracks, from more complex ones.

As it's not an heuristic function, some unrealistic movements can be generated. For example, the dummy user might appear in the middle of a lake, or crossing a highway while walking. Just by simple observation, the service provider or a malicious user can distinguish the real user from the dummy.
Also, we have to take into account that it's very likely that a real user realize frequently one or more itineraries with the same start and ending point (like going home to work, for example). Data mining of the different locations could lead to an eavesdrop of information, and to the revelation of the real geolocation.

Therefore, dummy generation algorithms that assure location privacy are a bit more complex, implying the use of some more elaborated techniques, like geocaching to avoid a possible data-mining [24], or the application of some Gaussian noise distribution function [21], due to the fact that even the x-y-z positions obtained by a GPS device are not 100% accurate, and have some implicit noise.

As in the previous cases, all this increase the time cost and network traffic, making this solution a bad approach to suit our needs.

The location privacy technique that will be developed in this work will be obfuscation. This kind of solution tries to protect the location of the user avoiding to reveal the real position.
With that objective, instead of sending an exact position p to the location server, a region P' is sent, where $p \subset P'$. The new region can be continuous, or discrete. To simplify the problem, we will assume a continuous region.

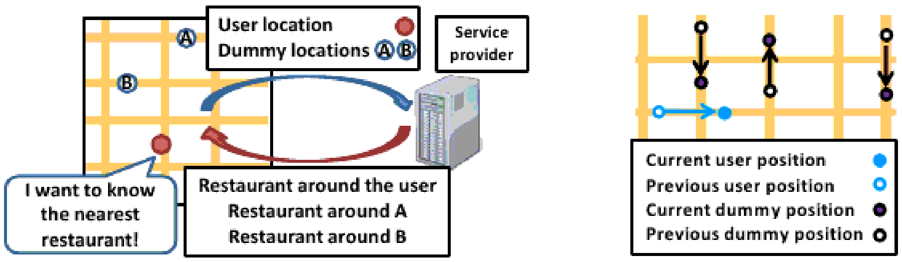


Figure 13: Service query and dummies generation [30]

Working with regions the quality of the information is degraded, reducing the accuracy and the quality of the service (QoS). Again, we found the classic trade-off between privacy and usability.

There are different alternatives [27] to deal with obfuscation:

a) **Obfuscation by enlarging the radius**:
Obfuscation by increasing the location of the measurement area is the most widely spread solution.

b) **Obfuscation by shifting the centre**:
Shifting the centre is another viable possibility. The new region can be derived by calculating the distance between the two centres. It provides the same privacy as the enlarging technique.

c) **Obfuscation by reducing the radius**:
It consists in reducing the radius of the location area. The privacy is produced by a correspondent reduction of the probability to find the real user location within the returned area.

Combinations between this techniques are possible. Although, there's no point in fusing more than 2 transformation. As we know, it's mathematically possible to transform any circumference in another one realizing at most two transformations. The combination between enlarging and reducing has no sense, but the Shift-Enlarge $S - E$ or Shift-Reduce $S - R$ double transformation are applied, and will be used.

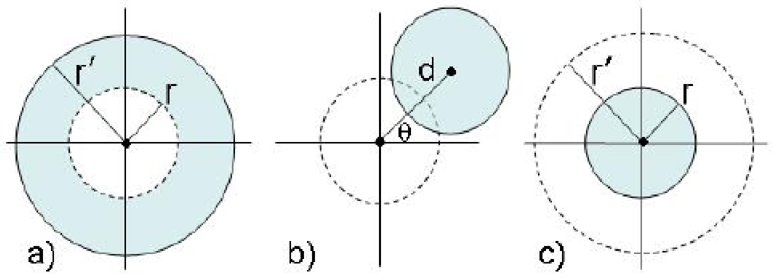


Figure 14: Obfuscation methods [27]

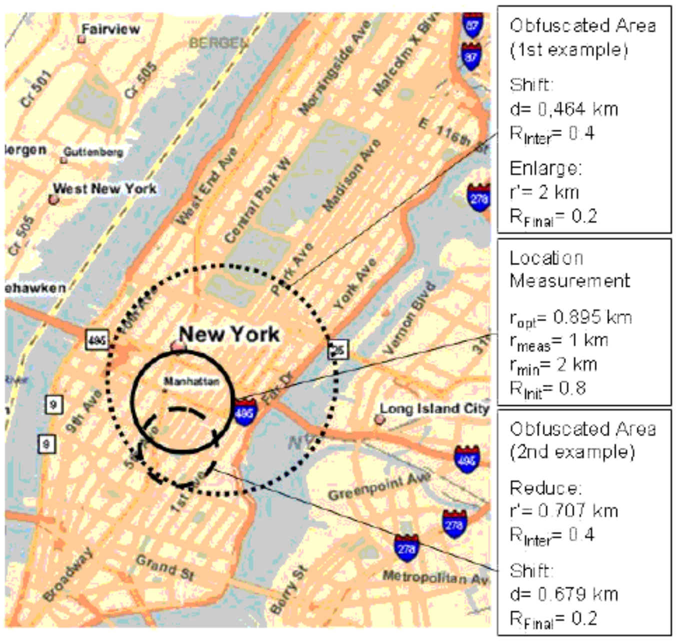


Figure 15: S-E and S-R transformation examples [27]

To measure the effectiveness of the different methods that try to assure location privacy (either unlinking the relationship between the user and her position, either hiding the real location), how to measure it should be defined. With that objective, different methods have been purposed. Among the most common ones, the relevance term [27] has to be defined. Relevance take into account the precision of the measure data, and the user preferences. It's not the same in terms of location privacy a user who does not want to reveal her position inside a radius of 20 meters, than other one whose radius is 10 kilometres.

Many of the concepts that have been explained here will be discussed and applied during this work.

3 Related work

3.1 Protecting location privacy against spatial inferences: the PROBE approach

Some concrete methods and approaches have been purposed in order to solve the location privacy problem.
One of this approaches, already mentioned in the previous work, is the PROBE approach.

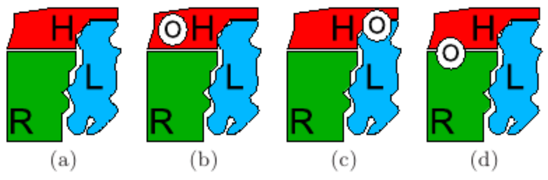


Figure 16: Different geographical context [31]

PROBE tries to protect the user's privacy but also taking into account the geographical context. For example, in the figure shown at the left of this paragraph, **H** represents *Hospital*, **L** stands for *Lake*, and **R** for *Residential District*, as seen in (a). Let's assume a user of the LBS to whom the Hospital is a sensitive location. Also, obfuscation will be assumed, increasing the region where the user can be. What would a malicious user be able to infer? We have different possibilities:

- (b) The obfuscated region is located inside the hospital: the attacker can easily deduce that the user is inside a sensitive area.
- (c) The obfuscated region is between the lake and the hospital: as it's impossible for the user to be inside the lake (as assumption), again, the location inside a sensitive area can be easily deduced.
- (d) The obfuscated region is between the hospital and the residential district: since the hospital is the only sensitive place, the attacker could say that the location is sensitive to some extent.

This leak is due to the fact that obfuscation techniques can't protect the user against the linking between semantic location and geometric information. In order to do this, the PROBE approach defines the sensitive metric, to calculate "how sensitive a region is". With that objective, the probability density function *pdf* is used.

The formula

$$P(r) = \int_r pdf \tag{3.1}$$

denote the probability that a user known to be in Ω is actually located in region r. If $P(r) = 0$, we say that the region is unreachable.

Therefore, the sensitivity of r with respect to the feature type Hospital can be defined as follows. Note that the sensitivity to any unreachable region is set to 0.

$$P_{sens}(Hospital, r) = \frac{Area(H_1 \cap^s r) + Area(H_2)}{Area(r \setminus^s L)}$$

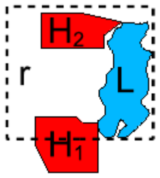


Figure 17: PROBE sensitivity function [31]

Once we have defined the sensitive metric, the obfuscation algorithm can be applied. It will use Hilbert curves to generate the obfuscated regions. Different algorithms are available [31].

These algorithms are applied to a single cell, and not to the whole sensitive feature. In this method, the original sensitive method is fragmented, using the locality propriety of Hilbert curves, and each fragment is expanded separately. Doing this, we can generate obfuscated locations that are smaller than the regions obtained obfuscating the entire region.

Applying this algorithm we obtain the desired obfuscated regions:

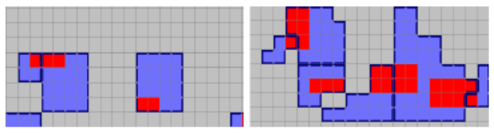


Figure 18: PROBE: generated obfuscated regions [31]

All locations on the map are represented as features. Each feature can has its sensitivity level, and therefore, it would be easy to apply these techniques.

3.2 Privacy preserving through a memorizing algorithm

One of the new attack methods to infer location information of a user is data-minning. It's very likely that an user realizes daily the same itinerary (going from home to work, from work to school to pick up the kids, and so on). Let's assume that obfuscation using dummies is being used. The dummies generated in every track will be different. Therefore, if an attacker is storing every location we send to the server, including the dummy ones, after some or many observations, it would be possible to infer the user location tracking, and distinguish it from the dummies.

To solve this,Quynh Chi Truong, Anh Tuan Truong, and Tran Khanh Dang [23] propose the following solution. The obfuscation algorithm has to memorize the rectangles, using a database. Therefore, at different times, the middleware will check the database in order to find it, and will save it if it hasn't been found. When the user wants to use the location service, she will send the real location to the middleware. The middleware will check if it's the first time. If yes, it will create the rectangle according to the privacy settings. Otherwise, it will get the rectangle from the database.

The pseoducode to do that is:

```
if (first time) {
  get random rectangle covering the user's
  position based on the privacy level;
  save this rectangle and the privacy
  level;
  return this rectangle;
}
else {
  if (less level){
    perform dividing function and get a
    proper rectangle;
    save this rectangle;
    return this rectangle;
  }
  else if (greater level){
    get the greatest saved rectangle;
    add some cells to this rectangle to
    satisfy the privacy level;
    save the added rectangle and the
    privacy level;
  return the added rectangle;
  }
  else { //equal level
    return the saved rectangle;
  }
}
}
```

We have to take into account that if an attacker knows the maximum speed of the user, and when does the user move out of a rectangle, the size of the area that contains the user can be limited. To avoid this, the authors propose a “time delay factor”, that means, the creation of the new rectangle would be delayed in time, to make the inference of any location information more difficult.

In order to guarantee privacy when using data-mining, these paper could lead to the use of a GPS database in order to implement the presented method. Because some of the techniques presented in this work make use of traffic knowledge, it would be possible to implement this solution, and therefore, it should be taken into account.

3.3 Preventing velocity-based linkage attacks

Cloaking regions (CR) are often used to provide location privacy. In fact, in this work, a CR will be purposed in order to solve privacy issues. As we know, at timestamp t_1, the user will send the region CR_1 instead of the real position. At timestamp t_2, the new region CR_2 will be send to the location server. The difference between t_i and t_{i+1} doesn't need to be the same for every i. If an attacker knows the maximum speed of the user (either because she's walking, riding a bike, or going by car), he could calculate the reachable region from any point in CR_1, and violate the location privacy.

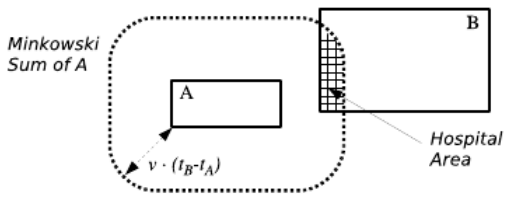


Figure 19: Breaching location privacy using velocity information [41]

The solution from [41] purposes a new algorithm to solve this linkage attacks.

Let's consider A to the CR_A and B to the CR_B. The metrics used to measure distances are the followings:

- **Hausdorff distance**: It's the largest distance between any point in A to some point in B.
- **Point-pairwise distance**: It's the maximum distance between any point in A to any point in B.

This metrics will be used for the explained methods.

The different methods an attacker could use to violate user's privacy depend on the knowledge of the attacker of the features in the map the user is in:

- The attacker doesn't have any map knowledge: the privacy objective is to prevent the disclosure of exact positions.

- The attacker has map knowledge: in this case, the attacker can access to the map features, this means, sensitivity of buildings can be used in the attack. We must ensure that the user can be located outside sensitive areas at both timestamps.

In this paper we're focused on map-awareness, and how it is achieved. Not just the attacking techniques, but also the disclosure conditions depend on this knowledge.

There are two different cloaking techniques that can provide the required conditions:

- **Temporal cloaking**: Temporal cloaking is suitable when the partition of the map into CRs is fixed in advance. Note that, since no computation is performed CR on-line, temporal cloaking is particularly suitable to be performed directly on the mobile device. As an additional benefit, performing cloaking on the device itself can make use of supplementary information about the user's trajectory. When the conditions mentioned above do not satisfy, instead of disclose the CR, this operation is delayed until the next request. There are different methods in how to implement this delay, like deferral and post date, but an heuristic between these two methods is the best approach to do so.

- **Spatial cloaking**: When the user's mobile device has sufficient resources, or when cloaking is performed by a trusted service, CRs can be dynamically computed at the time of the request. The advantage of such an on-line approach is that the CR can be tailored for the user's privacy profile, and consequently the QoS can be improved. A CR construction algorithm can take into consideration the boundary, and find a CR that is safe to disclose.

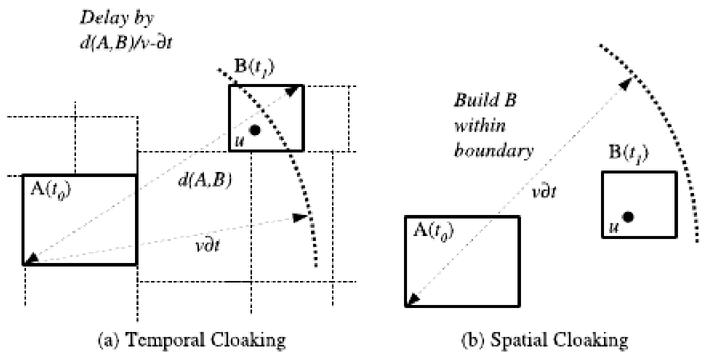


Figure 20: Different cloaking techniques [41]

In addition, the quality of the service shouldn't be reduced.

This work is interesting to our study thesis topic, due to the fact that shows the linkage attacks that an attacker could make when having knowledge of the features of the map, which is our case.

3.4 Landscape-aware location-privacy protection

The authors of [44] purpose a generic formalization to represent the associated problem when representing the different probabilities that can be covered by a cloaking region.
To do so, the following scenario is suggested: first of all, there will be an user A, who wanders within some landscape. This system also has a trusted agent, who is the one in charge of releasing the obfuscated location information to the third party. The trusted agent won't receive incorrect or wrong information, thus, he can receive non-precise locations, as result of executing obfuscation techniques and algorithms. Finally, there should be a third part agent, who will try to de-obfuscate the location information he receives.

In most real situations, landscape information can help the third party to perform inference over the data obfuscated, violating user's location privacy requirements. Hence, the conditions within the location of the user can be unveiled without affecting the configuration of the privacy levels without affecting location privacy profiles of the users, can be formalized and defined.

As it has already been mentioned, the user will send to the trusted agent the location information in a non-precise way. To do so, he can use techniques related with cloaking regions, using circles of radius r, or squares with size length 2R, depending on the coordinate system and metrics.
If no associated information is available(this means, the localization distribution function has the same probability for every location). obfuscation methods are easier to apply.

The application is provided with a $p_S LA/A$ probability by the user, which is the corresponding one to the maximum likelihood level within he wants to be located.

Therefore, the general statement of the problem is defined by

$$Pr(A \;\&\; C \mid r) = \int_s \mathrm{d}t \; Pr(A \; at \; t \mid B) \times \int_{\Omega_(t,r)} \mathrm{d}z \; Pr(C \; at \; z \;\mid A \; at \; t) \leq \mathbf{p_{SLA/A}} \tag{3.2}$$

where $Pr(A \;\&\; C \mid r)$ is the probability that the position of A is C within a distance r, S refers to the coordinate reference system, $\Omega_(t, r)$ is a region around

point t of radius r, t is the position of A, and z the position where C expects to find A.
In this paper the authors present different methods and conditions to reveal (or not) the imprecise location information, depending on the landscape scenario, and if it's neutral or not. Among this work this will be taken into account: if the landscape is non-neutral, the probability distribution of the user location results from the combination of the obfuscated information, and the information coming from the map.

Different probabilistic map representation techniques will be presented and studied in following sections.

3.5 Map-aware Position Sharing for Location Privacy in Non-trusted Systems

Dürr et al. [43] purpose an alternative method to guarantee location privacy. Location service provider (LS) might not be fully trusted by the users. Even if the servers are not malicious and misuse the data, they are susceptible to be attacked, which would compromise user's privacy. Therefore, instead of sending the precise position to the LS, this information is split in the so called position shares of limited precision. Using a share fusion algorithm, these shares can be combined to obtain a precise location. Doing this, different precision levels can be defined for individual applications.

Usually, the user might want to trade-off his privacy in terms of the precision of positions provided to location based applications (LBA) and the quality of service provided by the LBA. With this method, this could be easily done.

The presented scenario consist in a mobile object MO, location servers, and location based applications (LBA_i).

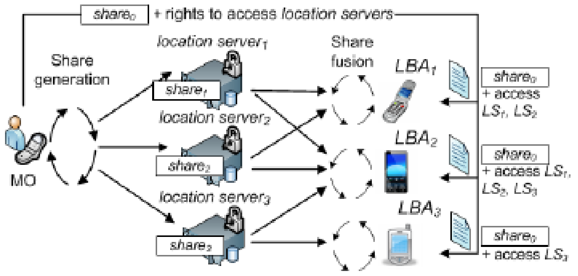


Figure 21: Map-aware position sharing model [43]

Giving a position p, the share generation algorithm produces a set of *shares* of minimum precision Φ_{min}: $generate(p, n, \Phi_{min}) = S$

This set $S = (s_0, s_1, ..., s_n)$ consists in a master position s_0 and a set of refinement shares.
Therefore, given a refinement s_k, a more precise position can be obtained using a share fusion algorithm: $fuse(so, s_k) = p_k$
Each refinement share increase the precision by a well defined value Δ_Φ. Fusing all the set reconstructs the precise position of the user.
Supposing map-awareness, and sporadically triggered share generation, this papers purposes three different algorithms which fulfils the presented properties:

- **Basic approach: open space**
  Positions are defined as circles with a determined radius (precision). Each

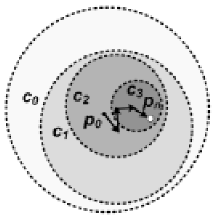


Figure 22: Open-space algorithm [43]

  refinement share defines a shift vector that shifts the center of the previous obfuscation circle. At the same time, the radius is decreased.
  In this approach, shares can be added in any order since the shift operation is a commutative operation. The maximal length of shift vectors has to be limited. As result, any refined circle is completely contained in the previous circle.

- **Extended approach: map-aware**
  Previous approaches suppose no knowledge of the map where the user

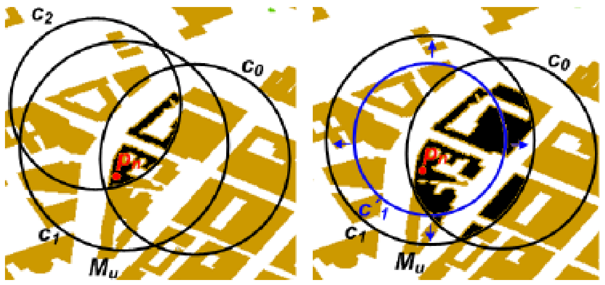


Figure 23: Map-aware algorithm [43]

  is in. When map-awareness is achieved, and attacker could increase his

radius of knowledge ignoring the areas where the user can't be (i.e, rivers, lakes, sea...) which will produce a privacy leak.
However, the solution to this is very simple: by increasing the size of the area of the location region($S = S_area - S_unlikely$), the algorithm presents the same properties already mentioned.

Using different security metrics, the quality of service of these algorithms has been evaluated, showing that the presented algorithms are a good and valid approach to protect location privacy.

This thesis will focus on this paper. In fact, many of the functions and algorithms that will be implemented and explained in this work could be used to implement the different methods presented here. Intersections between surface, and calculate intersected areas is needed as part of the map-aware algorithm. This study thesis will focus on that part of the algorithm.

4 Problem formulation

In the previous sections, we've studied the different formats available for representing data in geographic information systems. We considered also different methods and algorithms used to assure location privacy.
In the present work, as it has already been mentioned, we will be working with data in the shapefile format.This will lead to an easy integration with the previous work [12].
We will have as input the user location. In a future, this information will be obtained from a GPS sensor, but in this solution the position is obtained when the user clicks on any point of the map. Therefore, there's no need to take into account possible error measures due to the sensor, despite some studies [40] do so.

We want to provide location privacy using the obfuscation strategy. Instead of disclosing the real and exact position of the user, a circular region where the user is will be revealed. The method used for obfuscation will enlarge the privacy radius.
As the PROBE approach has shown on section 3.1, we have to calculate which features are inside the obfuscation area, in order to determine its characteristics, associated probabilities, and to guarantee the location privacy, increasing the radius when necessary.

Privacy radius will be defined by the user's application.With the objective to realize test, and to calculate areas, this software will use a small radius, related with the zoom level that is being used. The size of this cloaking region has to be adjusted to guarantee the location privacy of the user.

Therefore, and taking into account all the data and factors above mentioned, the software we have developed will calculate the features who are inside the privacy region, either because they're partially or completely inside it. When possible, the surface of the features inside the privacy region will be calculated.

The algorithm used to calculate the intersections is simple, and will be described in detail in the implementation section (5.2.2). As first step, a bounding box will be generated, containing the features that are likely to intersect(because they're close to the user position). After that, and using GeoTools API, we will check if they really intersect or not, calculating the intersected geometry when they do. Once we have that geometries, a coordinate reference system change has to be done, in order to calculate the intersected surface.

The goal of this study thesis is summarized in the following input and outcomes:

- As input to our system, we will have the probability of a user to be in a certain location. This data can be given as a probability distribution function. We should be able to represent in our map these probabilities, and show them graphically. To do so, we are using shapefiles. The location position of the user will be obtained when any point of the map is clicked. Also, we can have some traffic database knowledge, that will be used to calculate probabilistic map representations.

- As outcome, a restricted map region has to be defined, to determine if the position is likely to be disclosed or not, and therefore, if the location is secure (in privacy terms). The intersection between the privacy region and the features will be calculated. Also will be a probabilistic map representation approach.

Additionally, we will be able to assign feature (or feature type) probabilities, and to normalize the distribution probability within this region. And adjustment of an obfuscation circle size depending on normalized probability distribution of the neighbouring area should be possible.

Therefore, there are two main aspects in this work:

- **Map-awareness:** Features need to be extracted from the map files. The shapefile chosen format can represent buildings and objects as features. Doing this, every single feature can be associated with a privacy sensitivity level, permitting any algorithm to consider it in many different ways when calculating a safe cloaking region. Therefore, it would be possible to have access to map features, and to define a threshold or sensitivity level associated with every type of feature. This would allow to the user configure a privacy profile, defining which features and at which threshold level should be disclosed or not, depending on the location of the user.

- **Calculating intersections:** One of the main task is to obtain the intersected surface between the cloaking region and the selected feature.

  Detecting and calculating these intersection is a complex process. In a generic way, it can be expressed with the following expression:
  Let $\mathbf{MP} = \{MP_1, MP_2, ..., MP_n\}$ the set of n multipolygons close to the cloaking region.
  As we know, every multipolygon MP is formed by a set of polygons. Let $\mathbf{MP_i} = \{P^i_1, P^i_2, ..., P^i_m\}$ the set of m polygons which form part of MP_i, and let's call CR to the cloaking region.

The set MP is disjoint if $MP_i \cap MP_j = \emptyset \qquad \forall i, j / i \neq j$

Assuming a disjoint set of multipolygons, we can define the intersection surface **I.S** as

$$\mathbf{I.S.} \equiv CR \bigcap MP = \sum_{i=1}^{n} CR \bigcap MP_i \tag{4.1}$$

where

$$CR \bigcap MP_i = \textstyle\sum_{j=1}^{m} CR \bigcap P_j^i$$

represents the intersection between the CR and a specific multipolygon. Generalizing this expression for non disjoint features, we obtain the expression

$$\mathbf{I.S.} \equiv CR \bigcap MP = \sum_{i=1}^{n} CR \bigcap MP_i - \Delta S \tag{4.2}$$

where $\Delta S = \sum_{i=1;j=i+1}^{i=n-1 j=n} MP_i \bigcap MP_j \bigcap CR$

Therefore, the intersection between a circle (our cloaking region) and another geometric figure, should be analyzed. In some particular sceneries, these can be easily expressed.

Two main sub cases can be defined: intersections when the polygon is convex, and intersection when the polygon is concave. This second type, however, can be expressed as a set of convex polygons, and work with the formulas of the first case.

There are different algorithms and theorems to calculate the mentioned intersections (Polygon clipping [42], Two Ears theorem), but they do not form part of this study thesis, and therefore, won't be analyzed in detail. Fortunately, the GeoTools toolkit we're working with provides a powerful source of functions to calculate surfaces and intersections.

Due to the nature of this algorithms, and the conversions needed between different representation systems, small numerical errors will be introduced. If S is the intersected surface, and S' the calculated intersected surface, this error can be defined as:

$$\Delta\varepsilon = \mid S - S' \mid$$

5 Implementation

The previous version of the software presented in [12] has to be extended in order to calculate the intersection between features and the cloaking region. To do so, the MapViewerFrame class has to be extended. Also, an auxiliary class Constants has been defined.

5.1 UML diagrams

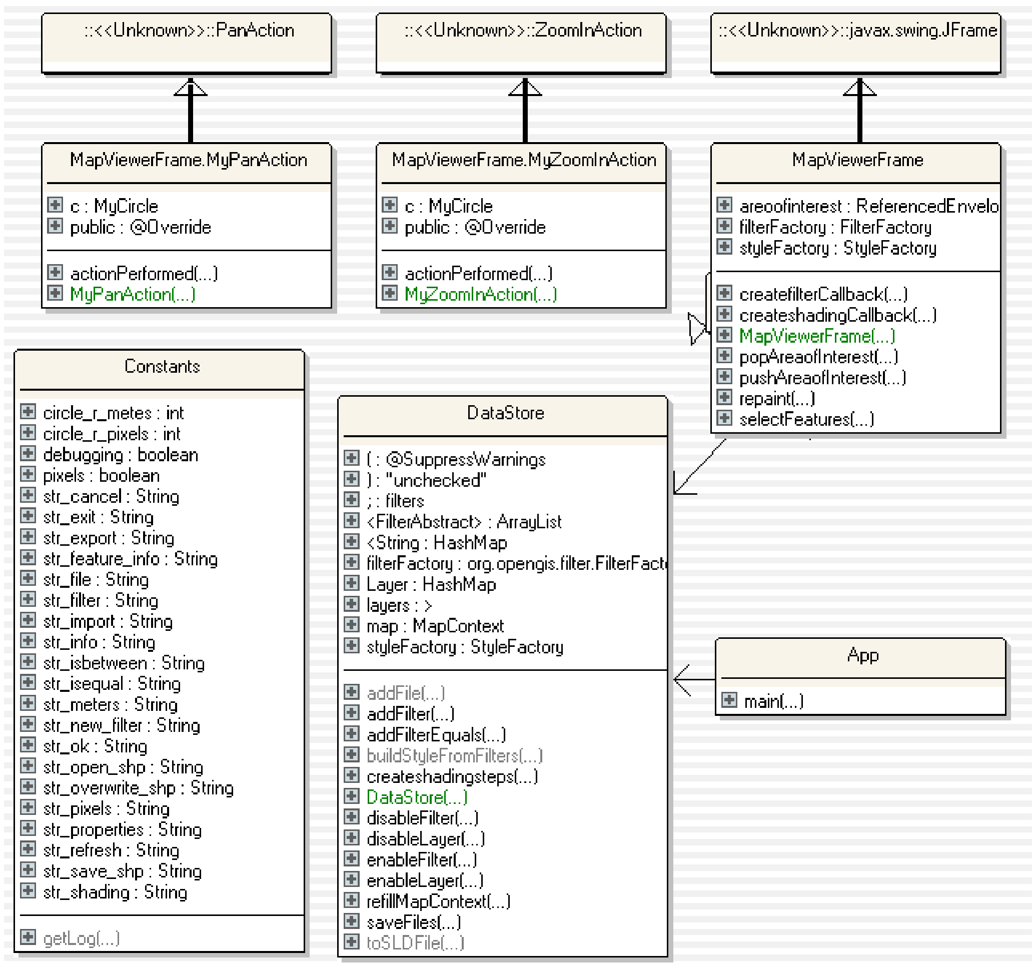


Figure 24: Global UML diagram

Global diagram The presented classes have been extended or defined to implement this part of the algorithm.

- **MapViewerFrame**: It has been extended in order to implement the intersections between features. It's explained in detail in the following sections.
- **Constants**: In order to make the code more easy to read and versatile. It's explained in detail in the following sections.
- **MyZoomInAction, MyZoomOutAction, MyPanAction**: extended classes of the ZoomAction and PanAction swing class, needed to paint the cloaking region.
- **MyCirlce**: extended class of swing JPanel, used to draw and represent the cloaking region.

Constants class

Now, we will focus on the constant class showed at the bottom left of figure 24. It's used to define some constants used in the program. The most interesting ones are:

- **Strings**: output messages to the user.
- **Boolean pixels**: if true, the size of the cloaking region is defined in pixels.
- **int circle_r_pixels**: size of the cloaking region radius in pixels.
- **int circle_r_meters**: size of the cloaking region radius in meters.
- **Boolean debug**: if true, a log output file will be generated.
- **String logfile**: route to the file where the log will be stored.
- **PMR PMR_method**: probabilistic map representation method.
- **int PMR_predefined_prob**: when using PMR with predefined values, if no value is set, this will be used.
- **PMR PMR_method**: probabilistic map representation method.
- **int cell_dim**: when using PMR cell method, it defines the NxN size of the window.
- **int cell_w**: when using PMR cell method, it defines the width of the window in meters.

MapViewerFrame class

A UML diagram of this class is shown in the upper right part of figure 24. In order to calculate the interested regions, the *s*electFeatures method had to be modified. The pseoducode is the following:

```
CR = new CloakingRegion(clickedPoint);
bbox = new BoundingBox)
intersection_list=new List();
for(int i=0;i<bbox.size;i++;){
  Polygon p=bbox[i];
  if p.intersects(CR)
    intersection_list.add(p);
}
calculateIntersectionSurface(CR, intersection_list);
```

The main part of the algorithm reside on *c*alculateIntersectionSurface. This method will call the different private functions in order to obtain the desired region.

The following auxiliary private methods have been defined:

- **String getWKTFromResource(String resName)**: receives a WKT file as input, returns the format. It's not used in the actual implementation, but might be useful in future versions.
- **String decimalFormat2(double d)**: formats a number to 2 decimal digits, in order to show it to the user.
- **String getRadio(Geometry g)** : given a cloaking region, return its radio.
- **Polygon constructPolygon(double height, double x, double y, GeometryFactory geometryFactory)**: creates a polygon in a determined geometryFactory. (x,y) is the upper left corner.
- **double calculateScaling(Point screenPos, AffineTransform screenToWorld,GeometryFactory geometryFactory)**: returns the actual scaling factor shown in the map. It's needed to represent the cloaking region in meters.
- **Geometry transformGeometry(Geometry g)**: transforms a given geometry to a different coordinate reference system.
- **double intersectedSurface(Geometry g1, Geometry g2)**: calculates the intersection between two geometries. It makes use of transformGeometry().

- **double calculateIntersectionSurface(SimpleFeatureCollection feat, Geometry circle)**: calculates the intersection between a list of geometries, and the cloaking region. It makes use of the intersectedSurface method.

5.2 Design

5.2.1 Point and its associated polygon

As we already know, the user can select any location of the map to choose a building (or another feature). If we want to calculate the surface of that building, we have to know which building (or features) are the ones who contain that point. A first approach to do so is asking every polygon if it contains the selected point.

As we're working with a high number of features, this solution doesn't seem to be a good method. Another alternative to do so would be using the Query Lab that GeoTools offers. This method works great for a spatial database, but there are better alternatives when we're working with shapefiles. One of the most common methods is to use a bounding box. If we treat the screen as a bounding box, and then we make a request for the content, it would be much faster, and there's no need to compare if all of the polygons contain that point, but just the ones inside the bounding box. More exacts polygon can be defined, using Filters. In fact, with this method, we can ask directly which polygon contains the point. As a different approach (but also very common), we can ask for the distance between the point and the polygons.

As it will be shown later, a comparative study has been done between the bounding box method, and some different filters. The less time-consuming one after the test for our case is a bounding box of a 3x3 pixel matrix. The pseudocode to do that is the following:

```
BoundingBox bbox=new BoundingBox(clickedPoint, 3, 3);
List l=new List(map.getPolygons(bbox));
while (l.hasNext() ){
 Polygon p=l.next();
 if ( Polygon.contains(clickedPoint) ){
   // Point inside polygon
   }
 }
```

5.2.2 Coordinate reference system

The coordinate reference system is an important part of this project. In order to calculate intersections, not only our privacy region, but also features, should be represented in the same coordinate reference system.
Unfortunately, there might be some lossy problems when converting between formats. Therefore, it's important to make this conversions only when needed.

Also, computationally speaking, they're high cost operation, so during the implementation, an effort in order to reduce the number of transformations has been done.

To do so, we have used the java advanced imaging core library (jai_core). By default, most of the maps in the shapefile format are stored using the world geodetic system (WG84). While this format is very useful in order to work with latitude and longitude coordinates, it's hard to work with it when calculating surfaces. Therefore, a transformation between coordinates system should be done. We're interested in working with a Cartesian system, in order to have the results in meters. The geomatics committee define many different formats (EPSG), and some of them suit our needs. We can transform our features either by indicating the EPSG code, or reading them from a file.
Unfortunately, and due to the format representation of the data itself, there might be some losses while doing this conversion. After some tests and calculating intersections between small buildings, the destination format chosen is DefaultGeocentricCRS.CARTESIAN.

To realize the conversion, we simply apply a Math transformation, suing the GeoTools reference system:

```
Geometry targetGeometry = null;
CoordinateReferenceSystem equalAreaCRS =
                                DefaultGeocentricCRS.CARTESIAN;
MathTransform transform = CRS.findMathTransform(
          DefaultGeographicCRS.WGS84, equalAreaCRS, true);
Geometry targetGeometry = JTS.transform(
          originalGeometry, transform);
```

5.2.3 Intersection of geometries

As part of our algorithm, we want to calculate if our privacy region (a circle of radius r) intersects with any of the features shown in the map. If so, we can calculate the intersected area.
As it has been already mentioned, the reference system play an important role here. It's important to know in which reference system is our privacy region represented, and in which reference system are our features being represented. When calculating the area, we can get incorrect results if we don't take this into account.

In order to reduce the transformation between coordinate reference systems, the steps to calculate intersections are the followings:

- **Calculate bounding box**: A reduced number of features will be inside the bounding box. Any feature that might intersect with the privacy region is inside it.

- **Calculate privacy envelope**: When calculating an envelope, it will use the same coordinate reference system as the maps we're working with. Therefore, no additional transformation would be needed.
- **Calculate intersected regions**: using the envelope, and the bounding box, we will check it the envelope intersects with any feature of the bounding box. If so, they will be added to an intersection list. The part of the source code which does this task is:

  ```
  if (circle.intersects(geometry)) {
    intersected_features.add((SimpleFeature) f);}
  ```

- **Calculate region surface**: Now the intersected list, if not empty, will be iterated, generating the intersected geometries, and transforming them to a Cartesian reference system. Doing so reduce the computational cost of the transformations. The surface of a geometry can be directly obtained using GeoTools API.

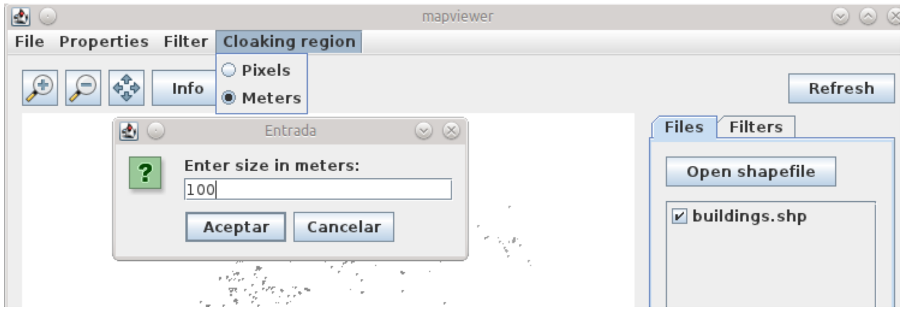
To simplify this process, and make the source code more human-readable, the following functions have been defined:

- **private Geometry transformGeometry(Geometry g)**: Receives a geometry, returns the transformed geometry.
- **private double intersectedSurface(Geometry g1,Geometry g2)**: Receives to geometries (in WG84 format), calculate the intersection, transform the resulting geometry, and calculates its area. It makes use of the transformGeometry method.
- **private double calculateIntersectionSurface(SimpleFeatureCollection feat, Geometry circle)**: Receives a list of features and the private region, returns the surface of the intersection between the circle and the features. It makes use of the previous functions.

Defining them, everything can be easily calculated just making use of calculateIntersectionSurface().

5.2.4 Defining radius

The length of the radius of the privacy region is defined directly into the source code. By default, it's defined by pixels (40 pixels), but it can easily be defined by meters (100 meters). Defining it by pixel size provide us the ability to change it's area depending on the zoom level: the higher the zoom is, the smallest the region.
We can simply change from one measure to another just selecting the option in the cloaking region menu.



Enter size in meters: 100

☑ buildings.shp

Figure 25: Adjusting CR size

To draw the circle region, the radius in pixel has to be specified. Therefore, in order to work with a CR specified in meters, some calculations have to be done. First of all, according to the scale, the pixels needed to represent a meter are calculated. According to this value, the radius of the circle(in pixels) is generated. As the scale might be too big, and pixels have to be specified as integers, some error ε could happen, but it will be irrelevant for our task.

The function which calculate this value is *p*rivate double calculateScaling(Point screenPos, AffineTransform screenToWorld, GeometryFactory geometryFactory). Then, the radius in pixel of the CR can be directly obtained:

```
boolean meters_mode=Constants.pixels;
if(!meters_mode)
  pixel_size=Constants.circle_size;
else
  pixel_size = (int) calculateScaling(screenPos,
                  screenToWorld,geometryFactory);
```

Everything can be easily adjusted by modifying the constants file:

```
public static boolean pixels;// default value: pixels(true)
public static int circle_r_pixels;    // radius, in pixels
public static int circle_r_metes;     // radius, in meters
```

5.3 Probabilistic map representation

One of the main tasks of this work is to represent different probabilities of users, in order to calculate better cloaking regions, and ensure privacy location. We can distinguish between different cases when doing probabilistic map representation (PMR), depending on if there's some traffic knowledge i.e. distribution function of the locations:

- If we don't have a traffic database, then objects should be checked:
  - If we don't have objects by itself, object types (buildings, for example), will be used. The same probability will be assigned to each of the object types.
  - If we have objects, a predefined probability of them will be used. How to assign it will be explained later.
- If we have a traffic database, then we should decide how we will be working:
  - If we work with cells (opensapce), the map will be divided in a set of cells. For each cell, the number of traffic points inside it should be counted. Doing this, we will be able to assign a probability to each cell.
  - If we work with objects, the number of traffic points inside each object should be counted, in order to assign a different probability to the different objects.

The next flow diagram summarizes the different cases we have. More accurate approaches use a traffic database. In concrete, when using objects, we're using also map knowledge, so it would be the best approach. The more generic ones don't use any traffic database. Assigning the same probability to every object type is the most simple solution, but also the most unrealistic.

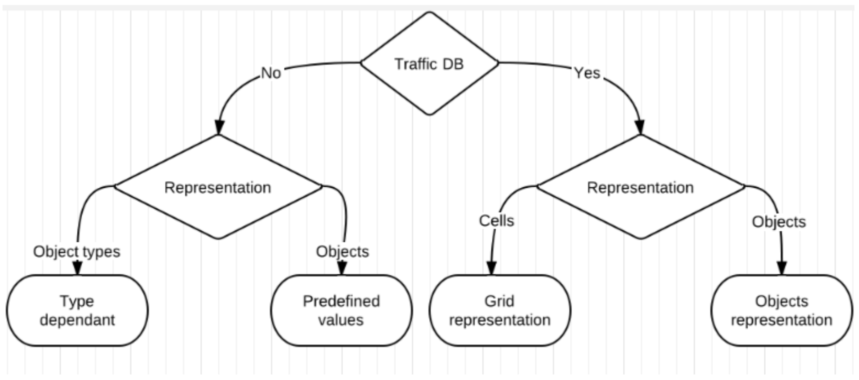


Figure 26: Probabilistic map representation: flow diagram

Any of the explained methods can be easily chosen from the menu. The size for the cell method can be defined. Also, these methods can be selected directly using the Constants file.

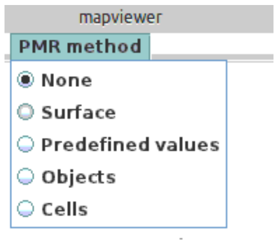


Figure 27: PMR: selection menu

5.3.1 Normalization

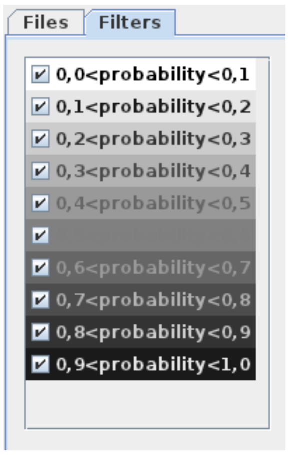
Once we have calculated the probabilities using one of the previous method presented, it can be useful for future calculations to normalize them. In order to do this, an external Statistics lib (com.aliasi.stats.Statistics) will be used. The usage of it is very intuitive: as input, it receives a vector containing the list of probabilities, and it produces a vector with the normalized probabilities as output:

```
array_prob = new double[n];
array_prob_N = new double[n];
array_prob = ...
array_prob_N = com.aliasi.stats.
            Statistics.normalize(array_prob);
```

5.3.2 Probabilistic graphical representation

Although probabilities and normalized probabilities can be consulted by checking the output or the logfile, it might be useful to have some kind of graphical representation.
A *s*hading filter can be used for this purpose, using the probability as parameter.

For the methods where this is not possible, like in cell PMR, a colouring function can be implemented manually.
It's important to notice that intersections are represented in red.



- ✓ 0,0<probability<0,1
- ✓ 0,1<probability<0,2
- ✓ 0,2<probability<0,3
- ✓ 0,3<probability<0,4
- ✓ 0,4<probability<0,5
- ✓
- ✓ 0,6<probability<0,7
- ✓ 0,7<probability<0,8
- ✓ 0,8<probability<0,9
- ✓ 0,9<probability<1,0

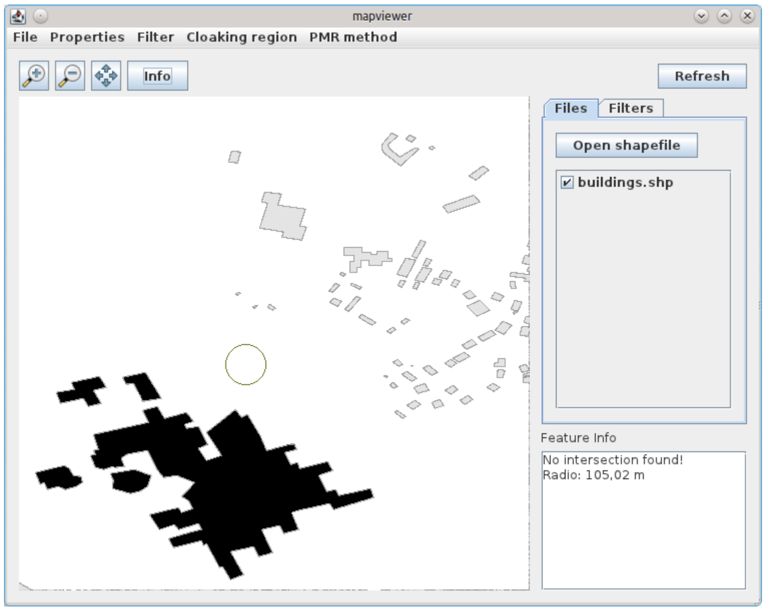
Figure 28: PMR: scaling probabilities

5.3.3 PMR: type dependent

For this representation, we will be working with a concrete object type (buildings, for example). Other kind of objects types won't be taken into account in the presented formulas.
Continuing with the schema shown on figure 26 ,let's suppose that we don't have any traffic knowledge, and also, that we don't have any objects defined, just object types (buildings). As there's no other value that could lead us to assign probabilities, we have to take the assumption that every object type represented has the same probability.
To do so, we will use a window (in this case, 200x200 px), of size S_w, and for every object type we're interested, we will get its surface. The probability of each object can be represented as:

$$P(obj_i) = S(obj_i)/S_w \tag{5.1}$$

In the following example, we can appreciate the list of calculated probabilities.
The objects types we're interested(buildings) are shown in grey. Other types of objects(lakes, parks) are shown in black. As explained before, they can be coloured using filters.



No intersection found!
Radio: 105,02 m

Figure 29: Probabilistic map representation: relative values

5.3.4 PMR: predefined values

Let's suppose that we don't have any traffic knowledge, but, this time, we have objects defined individually. These objects can be associated with some predefined probability values. Following the implementation of the previous study thesis [12], these properties can be imported from an external xml file.

In order to show how this works with an example, the probabilities of each object will be defined. In this case, a $P(obj|type = university) = 0.8$ will be set. The rest of objects will follow the rule $P(obj) = 0.001$:

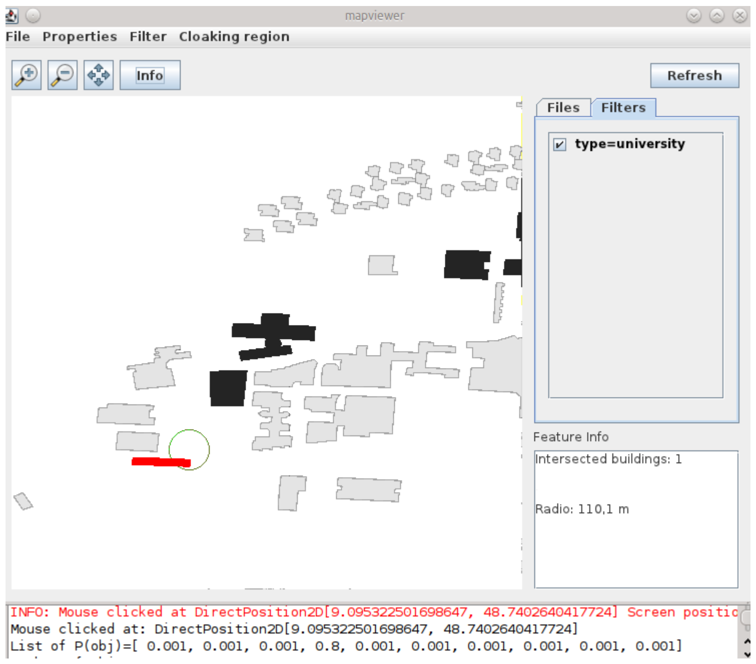
```
<?xml version="1.0"?>
<Properties>
  <Definition>
    <Attribute key="proba" type="Double" />
    <Attribute key="test" type="String" />
  </Definition>
  <set >
    <Filter>
      <PropertyIsBetween>
        <PropertyName>osm_id</PropertyName>
        <LowerBoundary>
          <Literal>0</Literal>
        </LowerBoundary>
        <UpperBoundary>
          <Literal>99999999</Literal>
        </UpperBoundary>
      </PropertyIsBetween>
     </Filter>
     <Attrib key="prob" value="0.001" />
  </set>
  <set >
    <Filter>
      <PropertyIsEqualTo>
      <PropertyName>type</PropertyName>
        <literal>university</literal>
      </PropertyIsEqualTo>
    </Filter>
    <Attrib key="prob" value="0.8" />
    <Attrib key="test" value="uni_building" />
  </set>
</Properties>
```

When trying to access the defined properties, the following java sentence should be used:

```
Feature f = ... ;
Double p=(Double) f.getProperty("prob").getValue();
```

This instruction gets the value of the property “prob”.

The following image shows the result of defining the associated probabilities as mentioned. University buildings are coloured in dark grey, and have an associated probability of $P = 0.8$. The rest of buildings are coloured in light grey, and have an associated probability of $P = 0.001$.
A list with the associated probabilities is shown below. These calculations are done as part of the getting intersected features process. The objects inside the



```
Intersected buildings: 1

Radio: 110,1 m
```

```
INFO: Mouse clicked at DirectPosition2D[9.095322501698647, 48.7402640417724] Screen positic
Mouse clicked at: DirectPosition2D[9.095322501698647, 48.7402640417724]
List of P(obj)=[ 0.001, 0.001, 0.001, 0.8, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001]
```

Figure 30: Probabilistic map representation: predefined values

window will also be represented with their associated probabilities, as shown in figure 30.

5.3.5 PMR: traffic knowledge, grid representation

Let's suppose that, this time, we do have traffic knowledge. The first task is to obtain and interpret this knowledge. To do so, we will make use of the KML format, already explained in previous sections. It represents the points in xml, therefore, it's easy to access to its contents and manage it using a parser.
An example of a valid KML file could be the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <TrackDB>
    <name>Simple track DB</name>
    <description>
      Distribution of points among the map
    </description>
    <Point>
      <coordinates>9.577081, 49.776003,0 </coordinates>
    </Point>
    ...
    <Point>
      <coordinates>9.577320, 49.775782,0 </coordinates>
    </Point>
    <Point>
      <coordinates>9.578148, 49.775152,0 </coordinates>
    </Point>
  </TrackDB>
</kml>
```

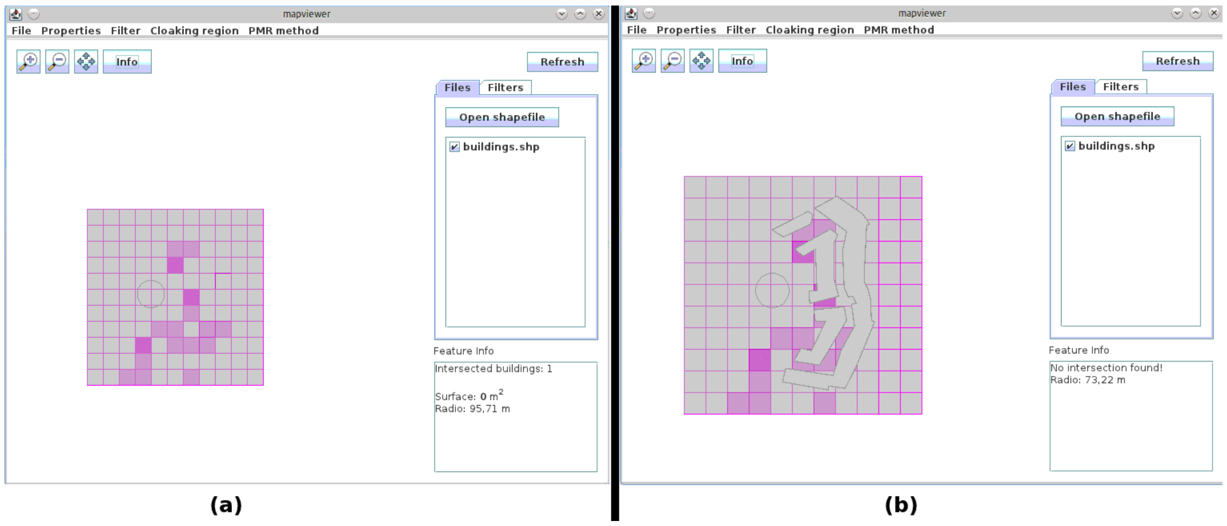
Once this file is defined and read, it's easy to work with a collection of points. Auxiliary functions have been defined in order to read the KML file, or check if an object contains a set of points.

Taking the assumption that no object is defined, the next step is to subdivide the window we're working with into cells. For testing purposes, a 220x220 window will be subdivided in 121 cells of 20x20.

Let's call N_w to the number of points inside the window, and N_i, j the number of points inside the cell C_i, j. The probability of an user to be inside a cell will be defined as

$$P(C_i, j) = \frac{N_i, j}{N_w} \tag{5.2}$$

In the following image, a matrix containing every C_i, j of the represented map is shown.



Intersected buildings: 1

Surface: **0** m^2
Radio: 95,71 m

No intersection found!
Radio: 73,22 m

Figure 31: PMR: a) Open space b) Open space with objects

Every single cell is coloured in a different way: if its probability is higher, its colour will be darker. In the software, only cells are represented. Here, we have overlaid cells and objects (although they're not used) to show exactly where the cells are created.

An example matrix representing the number of points is given by:

$$Cell_{i,j} = \begin{matrix} 0&0&0&3&2&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&3&1&0 \\ 0&0&0&0&0&0&0&0&1&2&0 \\ 0&0&0&0&0&2&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&2&0&0&0 \\ 0&0&0&0&0&0&3&2&1&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0 \\ 0&3&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \end{matrix}$$

The matrix containing the associated probabilities can be simply calculated as $\frac{Cell}{N_w}$:

$$P(Cell_{i,j}) = \begin{matrix} 0&0&0&\frac{3}{29}&\frac{2}{29}&0&0&0&0&0&0 \\ 0&0&0&0&\frac{1}{29}&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&\frac{3}{29}&\frac{1}{29}&0 \\ 0&0&0&0&0&0&0&0&\frac{1}{29}&\frac{2}{29}&0 \\ 0&0&0&0&0&\frac{2}{29}&0&0&0&0&0 \\ 0&0&0&0&0&\frac{1}{29}&0&0&0&0&0 \\ 0&0&0&0&0&0&\frac{1}{29}&\frac{2}{29}&0&0&0 \\ 0&0&0&0&0&0&\frac{3}{29}&\frac{2}{29}&\frac{1}{29}&0&0 \\ 0&0&\frac{1}{29}&0&0&0&0&0&0&0&0 \\ 0&\frac{3}{29}&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \end{matrix}$$

All the probabilistic work presented here is based on realistic approaches.

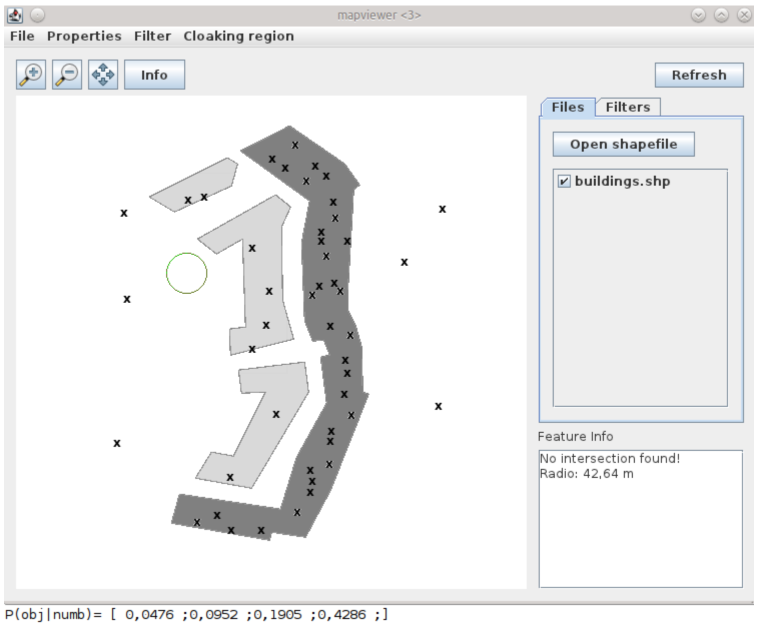
5.3.6 PMR: traffic knowledge, objects

This time, instead of using cells as basis, we will use the predefined objects.
To implement this method, a window will be defined, and every point inside the window will be counted. Let's call this number N_w. Once we have this number, the number of points that are contained in the objects inside the defined window will be counted. This number will be nominated as N_i, where i represents the building.

Not only the number of points inside it, but also the size of the object (S_i, in square meters) is indirectly taken into account, because only the points inside the intersection and inside the window are counted.
Working with this windows, the probability of an user to be inside the building is will be defined as

$$P(i) = \frac{N_i}{N_w} \tag{5.3}$$

In the following image, a table containing every P(i) for the given example is shown.



No intersection found!
Radio: 42,64 m

P(obj|numb)= [0,0476 ;0,0952 ;0,1905 ;0,4286 ;]

Figure 32: Probabilistic map representation: traffic knowledge and objects

Figure 32 also shows the location of different points contained in the track database. The object which contains more of this points will be, therefore, the one with the highest probability, and is represented in a darker colour.

5.4 Others

- **Constants class**: a new class (Constants) has been defined. It makes the code more easy to read, customize, and debug, many changes can be easily done just editing this class. Among others, Strings, radius value, and log file are defined here.
- **Read WKT (getWKTFromResources)**: as it has already been mentioned, the reference system can be read from a WKT file. In order to test different formats, a function to do so has been implemented:

  ```
  String getWKTFromResource(String file)
  ```

- **Debug**: a constant for debugging and a log file are defined now. When debugging, the trace of the program can be easily be checked just reading the log file.

6 Evaluation

To known how the presented algorithm works, different topics are evaluated on the following lines.
Also, some implementation aspects can be realized in different ways. In order to choose the solution which better suits the needs presented on this thesis, an analysis of different aspects has been realized.

In order to so, the following hardware has been used:

Device	CPU	RAM	Disk
Laptop	1.40 GHz	4096 MB	300 GB

Table 6.1: Hardware used for testing

6.1 Precision measure, transforming geometries, transforming coordinates

Precision measure:

Due to the conversion between pixels and meters, some precision error ε will be generated, as explained in 5.2.4. In order to evaluate it, different cloaking regions have been generated, with different scaling levels. The following table represent the difference between the real size of the cloaking region, and the calculated size:

Meters / **Zoom level**	**10**	**100**
Low	0.020	2.014
Medium	0.503	24.680
High	65.459	290.124

Table 6.2: Comparative table of precision error according to size and zoom level

As we can see, the higher the zoom is, the higher the error. Also, the error is correlated with the dimension of the radius. This is due to the fact that the scaling error is propagated.

Transforming geometries:

The transformation between different coordinate representation system can be done, either at the beginning, before calculating the intersection geometries, or at the end, meaning that the intersection is calculated first, and after that,

the result is transformed.
A priori, this second option needs less transformation operation, and therefore, should be faster. We will check it running a simple test between the directed and the delayed conversion algorithms, calculating the needed time to analyse a number of features. The processing time is represented in milliseconds:

Method / **Features**	**Direct**	**Delayed**
4700	2179	984
2000	480	253
150	169	138
10	1013	11

Table 6.3: Comparative table of different transforming geometries methods

As we said, if the transformation is delayed, much better results are obtained.

Coordinate reference system:

Depending on the Cartesian system used to represent our geometries, we can get better and more precise results. If we define our own reference system using an external WKT file, instead of using some of the default formats, we can define more parameters, and therefore, obtain better results. But, even though, measure errors will be introduced. As we don't want to affect the quality of service, a balance between processing time and measure error should be done. In order to select the best method, the transformation between coordinate reference systems using a default reference system, and one read from a file, has been realized. Time is expressed in milliseconds:

Method / **Features**	**File**	**Default**
1500	4055	239
25	123	22
10	72	23
2	1025	8

Table 6.4: Comparative table of different coordinate reference systems

Reading the format from a file highly increase the execution time, and the accuracy it can provide is more or less the same as any of the default reference systems. Therefore, WKT files won't be used.

6.2 Calculating intersected surface

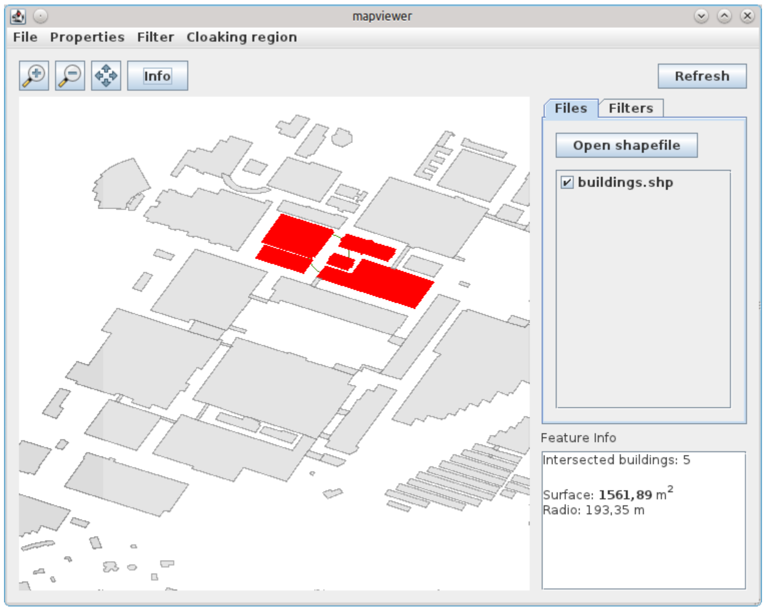
In order to evaluate how efficient and accurate our intersection algorithm is working, precise coordinates of the different features of buildings are needed.

As input, we only have the information stored in the map. Anyway, we can draw the cloaking region area, and check visually the number of features that are intersecting with our CR, in order to measure how good the developed methods work. Three different scenarios are presented:

- Buildings shapefile, no intersection:

  No intersection will be displayed or detected.

- Buildings shapefile, 6 intersections:



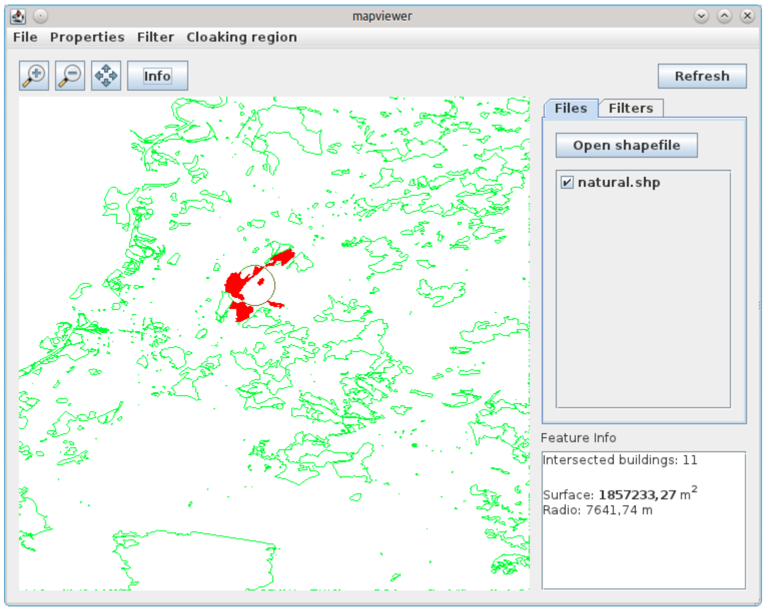
Intersected buildings: 5

Surface: **1561,89** m^2
Radio: 193,35 m

Figure 33: Program: intersections found (city)

The correct number of intersected features is displayed, and the intersected surface between the CR and the features is shown. The features which intersect with the CR are shown in red.

- Forest shapefile, intersection:



Intersected buildings: 11

Surface: **1857233,27** m^2
Radio: 7641,74 m

Figure 34: Program: intersection found (forest)

Again, the intersected features are shown in red. This time, the intersected surface is much bigger, as it's correctly displayed on the image.

Due to the data format of the shapefile, small calculating errors can be introduced. This means, that for small cloaking regions (5 meters radius), the calculated intersection surface might not be very accurate. Fortunately, we will rarely work which such cases, and when working with bigger surfaces, the algorithm behaves as expected, as seen on Figure 34.

6.3 Point and its associated polygon

To evaluate this search method, 10 random points inside a building have been selected, using different zoom levels. The numbers of comparisons needed to find the belonging polygon, and the time (in milliseconds) needed, have been measured, and an arithmetic mean has been calculated.

The methods compared are: check all polygons, 3x3 bounding box, 10x10 bounding box, intersect filter, and the point filter. The results are the following:

As we can see, the smallest the bbox is, the faster the method is. The bbox method also has a big correlation with the zoom level(the higher the zoom is, the less polygons inside the bbox are). Using a more exact polygon with the intersection method will result in a slower check, but less features will be

Method	**None**	**bbox(3x3)**	**bbox(10x10)**	**Intersect**	**Point filter**
Polygons	17490	1.2	10.5	**1.1**	**1.1**
Time(ms)	850.6	**5.9**	10.1	600.7	7.6

Table 6.5: Associate point to polygon techniques

retrieved. The point filter doesn't need a big amount of time to be realized, and also very few polygons need to be checked. Therefore, a small bounding box, or the point filter, are the two methods that better suit our implementation. As in order to calculate intersections, we might need not just the polygon who contains the clicked point, but also the ones close to them, the bounding box method is the one we will use.

6.4 Runtime

In order to measure runtime, a bounding box filter has been defined (as explained in the implementation section). After that, 10 different tests have been executed for each zoom level (granularity) and probabilistic map representation method.

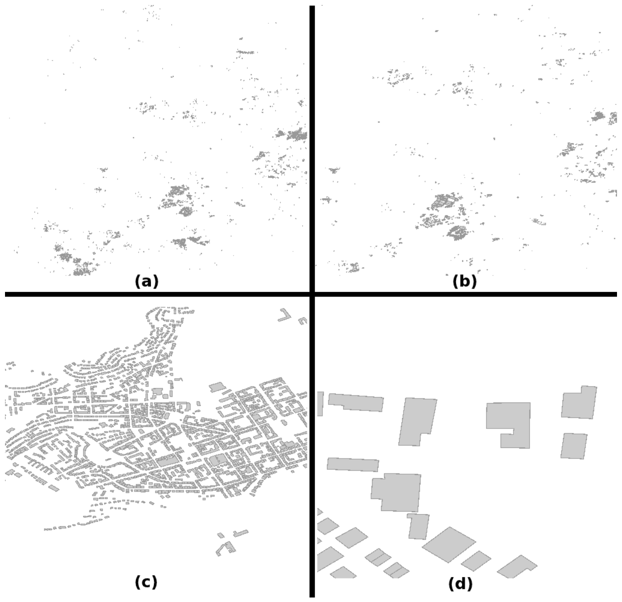


Figure 35: Different granularity levels: a) very coarse b) coarse c) fine d) very fine

The average of this 10 iteration has been calculated, and it's represented in the following tables:

Granularity	**Objects**	**Time(ms)**
very coarse	2662,7	1181,8
coarse	525,4	418,4
fine	34	38,3
ultra fine	2,1	19,8

Table 6.6: Running time, no PMR

Granularity	**Objects**	**Time(ms)**
very coarse	1868,2	9416,7
coarse	880,5	2229,4
fine	30,8	108,4
ultra fine	2,9	27,8

Table 6.7: Running time, surface PMR

As we can see in these tables, as we increase the zoom level, and therefore, we use a finer granularity, the number of intersections that need to be calculated is reduced, and therefore, the time needed to compute the different operations decreases.
When using a probabilistic map representation approach, the time (measured in milliseconds) is increased: more operations and instructions need to be accomplished in order to obtain the desired data.

In the following cases, when using a traffic database, the time used to load it is not taken into account.

Granularity	**Objects**	**Time(ms)**
very coarse	1406	24706,5
coarse	480,9	1548,3
fine	43,8	168,4
ultra fine	2,8	130,8

Table 6.8: Running time, predefined PMR

When using a predefined probabilistic map representation, the time needed is higher than the one we need when used surface-based PMR. This is because every object needs to be read an accessed in order to find the "probability" property.

Granularity	**Objects**	**Time(ms)**
very coarse	2113,1	14970,3
coarse	407,5	473,1
fine	61,4	146,9
ultra fine	2,1	130,3

Table 6.9: Running time, objects PMR

For cells, a 500 meters and 10x10 grid has been defined for every test. It's enough to our purposes, because the track database we have defined is not so big.

Granularity	**Objects**	**Time(ms)**
very coarse	2692	3983,5
coarse	556,4	2360,2
fine	58,5	1778,6
ultra fine	1,9	1728,2

Table 6.10: Running time, cells PMR

Not only in the objects PMR, but also in cells PMR, objects (either the buildings or the cells from the cell grid) need to be used in order to find the number of points inside it. The more objects we have (coarse granularity), the more operations need to be calculated. Therefore, the time will be higher. When using cells, we also have to define the objects(grid cell), therefore, it will be slightly slower than the objects PMR method.

6.5 CPU, disk needs, and memory usage

As many of this probabilistic map representation and location based services will be used in mobile devices, it would be useful to measure the results obtained in a smartphone device.
Unfortunately, up to this date, the geotools toolkit is not fully available for mobile operative system like android, iOS, or windows mobile, and to adapt this toolkit for this kind of operative systems lays out of the scope of this work. GeoTools toolkit is available however to windows and most of the UNIX systems, and to any device which can run java.

What can be done, however, is to measure some relevant factors like disk, CPU and memory usage, and to estimate how long will it take to realize the same operations in a smartphone device, taking into account the hardware resources available on it.

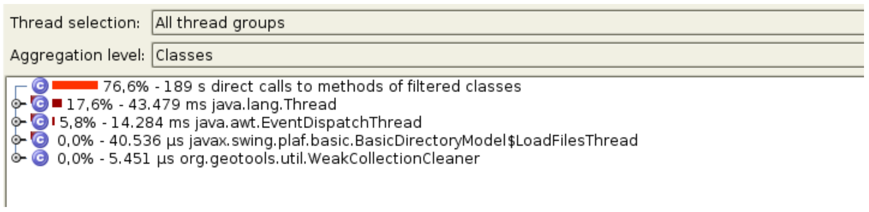
Let's assume that the previous test were realized in the following hardware.

In order to measure diverse metrics of CPU and memory usage, a tool called Java Profiler[45] has been used for this purpose. During the different test sets presented on the previous section, it has been used to calculate the average of CPU and memory usage of the program. The results obtained are the following:

- CPU usage: 76.6 % (1.07 GHz)
- Memory usage: 44.3 kB

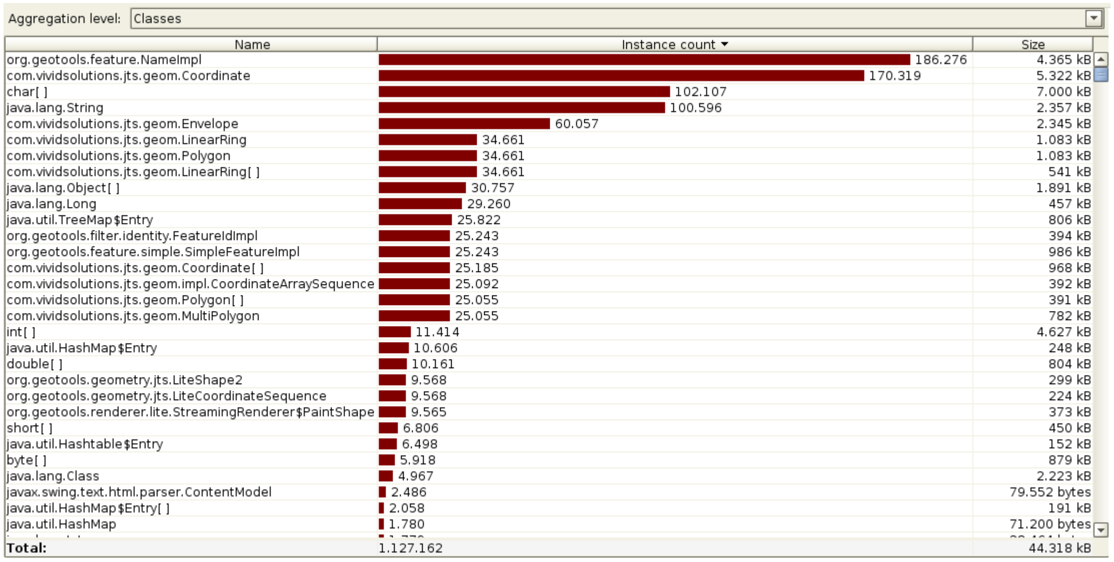
For the disk storage, let's suppose that our program has a set of 5 maps. Each map (in shapefile format) has a size of 165 MB. The jar file, with the geotools lib included, has a size of 50.3 MB. Assuming a track database file of 150 MB, and filters and properties files with less than 1 KB, the storage space needed would be: $156 * 5 + 150 + 50.3 \approx$ **980 MB** which is not a problem for actual devices.

The runtime of our algorithms will be limited then by CPU and memory storage. This will lead to non-usable algorithms when using coarse granularity. Fortunately, usually we will be working with fine granularity, and runtime will be also limited by network speed and connection.



```
Thread selection:   All thread groups
Aggregation level:  Classes
76,6% - 189 s direct calls to methods of filtered classes
17,6% - 43.479 ms java.lang.Thread
5,8% - 14.284 ms java.awt.EventDispatchThread
0,0% - 40.536 µs javax.swing.plaf.basic.BasicDirectoryModel$LoadFilesThread
0,0% - 5.451 µs org.geotools.util.WeakCollectionCleaner
```

Figure 36: CPU usage


Aggregation level: Classes

Name	Instance count	Size
org.geotools.feature.NameImpl	186.276	4.365 kB
com.vividsolutions.jts.geom.Coordinate	170.319	5.322 kB
char[ ]	102.107	7.000 kB
java.lang.String	100.596	2.357 kB
com.vividsolutions.jts.geom.Envelope	60.057	2.345 kB
com.vividsolutions.jts.geom.LinearRing	34.661	1.083 kB
com.vividsolutions.jts.geom.Polygon	34.661	1.083 kB
com.vividsolutions.jts.geom.LinearRing[ ]	34.661	541 kB
java.lang.Object[ ]	30.757	1.891 kB
java.lang.Long	29.260	457 kB
java.util.TreeMap$Entry	25.822	806 kB
org.geotools.filter.identity.FeatureIdImpl	25.243	394 kB
org.geotools.feature.simple.SimpleFeatureImpl	25.243	986 kB
com.vividsolutions.jts.geom.Coordinate[ ]	25.185	968 kB
com.vividsolutions.jts.geom.impl.CoordinateArraySequence	25.092	392 kB
com.vividsolutions.jts.geom.Polygon[ ]	25.055	391 kB
com.vividsolutions.jts.geom.MultiPolygon	25.055	782 kB
int[ ]	11.414	4.627 kB
java.util.HashMap$Entry	10.606	248 kB
double[ ]	10.161	804 kB
org.geotools.geometry.jts.LiteShape2	9.568	299 kB
org.geotools.geometry.jts.LiteCoordinateSequence	9.568	224 kB
org.geotools.renderer.lite.StreamingRenderer$PaintShape	9.565	373 kB
short[ ]	6.806	450 kB
java.util.Hashtable$Entry	6.498	152 kB
byte[ ]	5.918	879 kB
java.lang.Class	4.967	2.223 kB
javax.swing.text.html.parser.ContentModel	2.486	79.552 bytes
java.util.HashMap$Entry[ ]	2.058	191 kB
java.util.HashMap	1.780	71.200 bytes
Total:	1.127.162	44.318 kB

Figure 37: Memory usage

6.6 Mobile solution

Nowadays, GeoTools API is not available for the different mobile OS. Therefore, it's not possible to implement the presented algorithms in smartphones or mobile devices.

As it has already been mentioned, GeoTools is a toolkit developed in Java. However, with some slightly modifications, a certain number of libraries could be used in the Android platform [47]. Through an incompatibilities analysis, it should be possible to run the program shown in this work in Java compatible platforms, like JavaME, Blackberry, or Android. Those OS have an available emulator for PC, so there's no need to have a real device in order to implement and test the solution.
Unfortunately, this modifications lay out of the scope of this work.

OS	**Debugger**	**Language**
Android	✓[1]	Java[2]
Blackberry	✓[1]	Java
IOS SDK	✓[3]	Object Pascal
Palm OS	✓	C, C++
QT SDK	✓	C++
Windows Mobile	✓	C, C++

Table 6.11: Comparative of different mobile OS

Another alternative would be to implement this methods using a different library. Some of them are GeoTools and JTS [48] for Java, or GEOS [49] for C, all of them open source.

[1]Integrated with Eclipse
[2]Portions of code can be in C,C++
[3]Integrated with Xcode

7 Conclusions

At this work a study about different available data formats which can be used to represent geographic information has been realized. At the same time, the tools available for this task have been analyzed. More concrete, the GeoTools toolkit, including a study of the advantages and disadvantages of the representation data formats mentioned above related with their compatibility with this suite.

Once all this has been studied in context, different techniques used to provide location privacy have been shown. Specially, a concrete study of different obfuscation techniques has been realized.
The use of this techniques is strictly associated with intersections of surface computation. Therefore, this methods have been implemented as an additional functionality of the previous study thesis, which also includes computation of surfaces, and intersections of different features with the cloaking region.

Different methods for probabilistic map representation have been formalized, implementing them, and representing probabilistic values in a graphic way.

Runtime, number of intersections needed, and some other aspects, have been evaluated, in order to determine the feasibility of the implementation of this techniques in mobile devices. As presented in section 6.4 (Evaluation), although a higher runtime and memory consumption can be expected, it would be feasible to implement this methods for their usage in less powerful hardware (i.e smartphones and tablets).

In a future version, it should be possible to work not only with shapefile data format, but also with OSM. Also, a privacy profile could be defined, in order to extend the presented algorithms, and disclose (or not) the real position of the user. These methods could also be slightly modified to automatically adapt the cloaking region radius, or to make use of shifting center obfuscation techniques. The implementation of a real mobile application will deal to a more accurate evaluation of these methods.

References

[1] OSGeo Geotools, *Geotools documentation* `http://docs.geotools.org/`

[2] Canadian Geomatics Interchange Standard, *Spatial Archive and Interchange Format: Formal Definition.* Canadian General Standards Board, CGSB 171.1-95-CAN/CGSB, 1995.

[3] Canadian Council on Geomatics Interchange Format, *CCOGIF Quick facts* `http://docs.safe.com/fme/html/FME_ReadersWriters/FMEReadersWriters_Left.htm#CSHID=ccogif/ccogif_ascii_quick_facts.htm|StartTopic=ccogif/ccogif_ascii_quick_facts.htm`

[4] American National Standard Institute, *Spatial Data Transfer (Draft)* ANSI NCITS 320-1998, `http://mcmcweb.er.usgs.gov/sdts/standard.html`

[5] USGS, *Digital Line Grap Standards: DLG-3* U.S. Department of the Interior, 1996. `http://nationalmap.gov/standards/dlgstds.html`

[6] Open Geospatial Consortium, *Simple Feature Access: Common architecture* 2011. `http://www.opengeospatial.org/standards/sfa`

[7] GML, *Geography Markup Language* Encoding standard documentation, version 3.2.1, `http://www.opengeospatial.org/standards/gml`

[8] *OpenGIS Web Feature Service Specification* Open Geospatial Consortium. `http://www.opengeospatial.org/standards/wfs`

[9] Lance Alan Dyas, *GeoXML v3* June 2011, `http://code.google.com/p/geoxml/`

[10] *CityGML* Open Geospatial Consortium, 2011. `http://www.citygml.org/`

[11] ESRI White Paper, *ESRI Shapefile technical description.* Environmental System Research Institute, United States, 1998. `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`

[12] Andreas Paul, *Visualisierung von Kartenobjekten mit GeoTools* Universität Stuttgart, July 2011, `ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/STUD-2312/STUD-2312.pdf`

[13] OSM, *OpenStreetMap documentation* `http://wiki.openstreetmap.org/wiki/Main_Page`

[14] Topografix, *GPX Exchange Format* Version 1.1, August 2004, `http://www.topografix.com/GPX/1/1/`

[15] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, Chirstopher Schmidt. *GeoJSON Format Specification* 2008. `http://geojson.org/geojson-spec.html`

[16] Fosca Giannotti, Dino Pedreschi. *Mobility, Data Mining and Privacy* Springer, Pisa, Italy, 1st Edition, 2008.

[17] Beresford, A.R., Stajano, F. *Location privacy in pervasive computing* Pervasive Computing, IEEE, p46-55, 2003.

[18] Frank Dürr, Pavel Skvortsov, Kurt Rothermel. *Position Sharing for Location Privacy in Non-trusted Systems* PerCom, 2011.

[19] Hassan Takabi, James B. D. Joshi, Hassan A. Karimi. *A Collaborative K -anonymity Approach for Location Privacy in Location-Based Services* CollaborateCom, 2009.

[20] Emmanouil Magkos. *Cryptographic Approaches for Privacy Preservation in Location-Based Services: A Survey* 2011.

[21] Yan Huang, Roopa Vishwanathan. *Privacy Preserving Group Nearest Neighbour Queries in Location-based Services using Cryptographic Techniques* IEEE GlobeCom, 2010.

[22] Amir Salar Amoli, Mehdi Kharrazi, Rasool Jalili. *2PLoc: Preserving Privacy in Location-Based Services* Iran, IEEE, 2010.

[23] Quynh Chi Truong, Anh Tuan Truong, Tran Khanh Dang. *Privacy Preserving through A Memorizing Algorithm in Location-Based Services* MoMM, 2009.

[24] F. Diggelen. *Gnss accuracy: Lies, damn lies, and statistics.* GPS World, 2007.

[25] Man Lung Yiu, Christian S. Jensen, Jesper Moller. *Design and Analysis of a Ranking Approach to Private Location-Based Services* May 2011.

[26] A.J. Bernheim Brush, John Krumm, James Scott. *Exploring End User Preferences for Location Obfuscation,Location-Based Services, and the Value of Location* SIGCH Conference, Microsoft, 2011.

[27] C.A. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, P. Samarati. *Location Privacy Protection Through Obfuscation-based Techniques* 2007.

[28] Andreas Paul. *Visualisierung von Kartenobjekten mit GeoTools* 2011.

[29] Marco Gruteser,Xuan Liu. *Protecting Privacy in Continuous Location-Tracking Applications* IEEE Security, 2004

[30] Akiyoshi Suzuki, Mayu Iwata, Yuki Arase, Takahiro Hara, Xing Xie, Shojiro Nishio. *A User Location Anonymization Method for Location Based Services in a Real Environment* Microsoft Asia, 2010.

[31] Maria Luisa Damiani, Elisa Bertino,Claudio Silvestri. *Protecting location privacy against spatial inferences: the PROBE approach* 2009.

[32] OpenLayers documentation, *http://trac.osgeo.org/openlayers/wiki/Documentation*

[33] U.S. Census Bureau, *http://www.census.gov/*

[34] A. J. Bernheim Brush, John Krumm, James Scott. *Exploring End User Preferences for Location Obfuscation,Location-Based Services, and the Value of Location* Microsoft Research, 2010.

[35] Subsquehanna River Map gallery, *http://www.srbc.net/gis/map_gallery.html* 2007.

[36] 3D Analyst extension suit, *http://www.esri.com/software/arcgis/extensions/3danalyst/index.html* ArcGIS.

[37] OpenStreetMap-3D, *http://www.osm-3d.org/home.en.htm*

[38] Ruprecht-Karls-Universität, *Abteilung für Geoinformatik* Heidelberg. `http://www.geog.uni-heidelberg.de/lehrstuehle/gis/`

[39] RouteConverter, 2007. `ww.routeconverter.de`

[40] Pravin Shankar, Vinod Ganapathy,Liviu Iftode *Privately Querying Location-based Services with SybilQuery* Rutgers University, 2009.

[41] Gabriel Ghinita, Maria Luisa Damiani,Claudio Silvestri *Preventing Velocity-based Linkage Attacks in Location-Aware Applications* SIGSPATIAL International Conference , 2009.Landscape-aware location-privacy protection in location-based services

[42] Patrick-Gilles Maillot *A New, Fast Method For 2D Polygon Clipping: Analysis and Software Implementation* Sun Microsystems, 1992

[43] Pavel Skvortsov, Frank Dürr, Kurt Rothermel *Map-aware Position Sharing for Location Privacy in Non-trustedSystems*, Proceedings of the 10th International Conference on Pervasive Computing (Pervasive 2012), June 2012, Newcastle, UK.

[44] Claudio Agostino Ardagna, Marco Cremonini, Gabriele Gianini *Landscape-aware location-privacy protection in location-based services* Journal of System Architecture, p243-254, 2009.

[45] EJ technologies, *Java Profiler documentation manual* 2012. `http://resources.ej-technologies.com/jprofiler/help/doc/`

[46] Bugra Gedik *Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms* IEEE transactions on mobile computing, vol. 7, January 2008.

[47] Atlassian Confluence Open Source Project, *GeoTools for Android* November 2011. `http://docs.codehaus.org/display/GEOTOOLS/Android`

[48] Vivid solutions, *JTS Topology Suite* `http://www.vividsolutions.com/jts/JTSHome.htm`

[49] Edgewall software, *Geometry Engine Open Source* `http://trac.osgeo.org/geos/`

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Daniel del Hoyo)