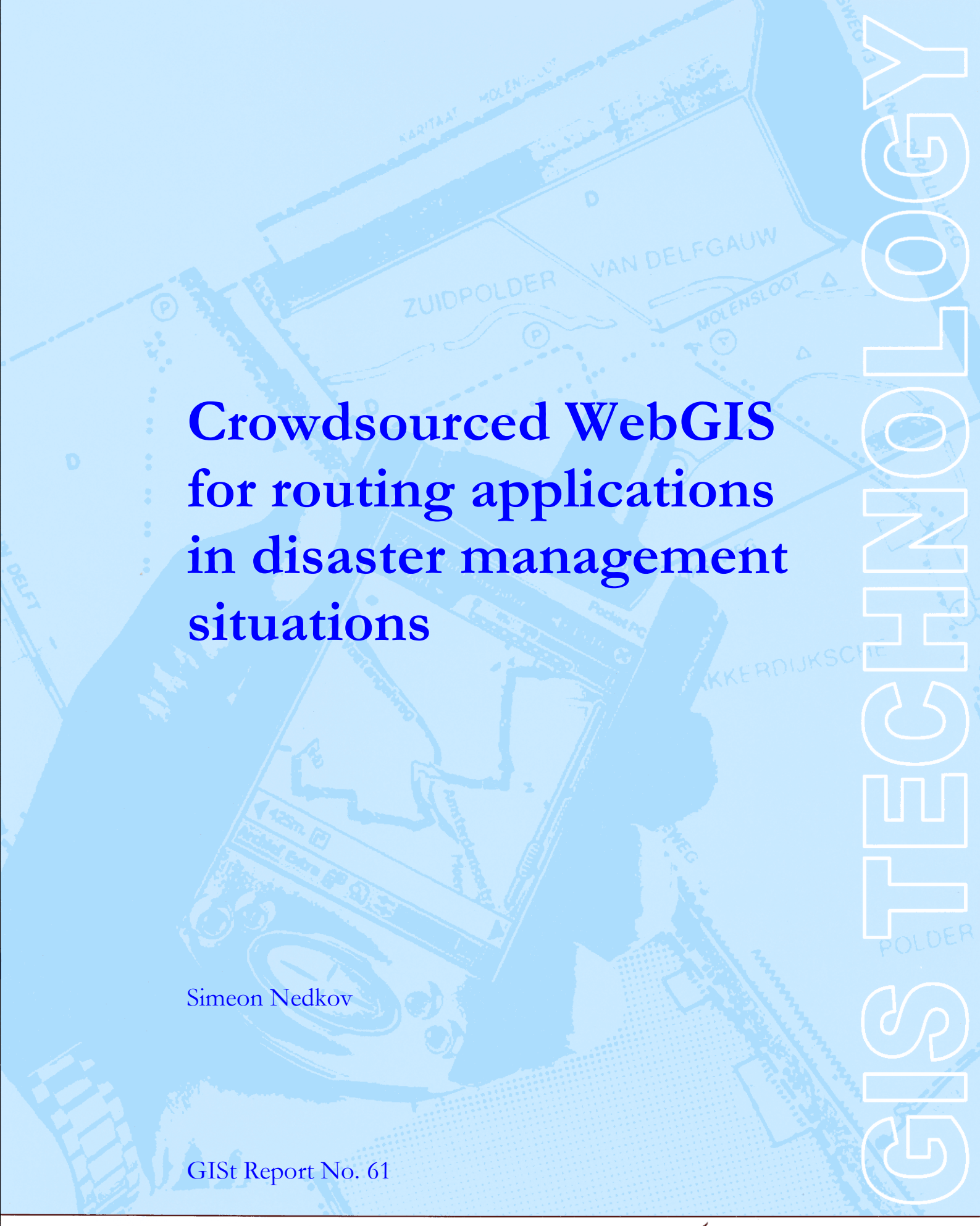

GIS TECHNOLOGY

Crowdsourced WebGIS for routing applications in disaster management situations

Simeon Nedkov

GISt Report No. 61

TU Delft

OTB Research Institute for Housing, Urban and Mobility Studies

Delft University of Technology

Crowdsourced WebGIS for routing applications in disaster management situations

Simeon Nedkov

GISt Report No. 61

Summary

Successfully navigating a damaged infrastructure is challenging due to a lack of automatic routing solutions and a shortage of real-time infrastructure status information. Remote sensing techniques, which are traditionally used, have drawbacks; clouds obscure the view, observation frequency is low, many phenomena can only be observed from the ground, etc.

This report presents an alternative observation strategy in the form of crowdsourcing: untrained volunteers are engaged in observing the state of the infrastructure. A web application is built that enables volunteers to make observations through desktop and mobile devices and use the collected information to plan the shortest route to a certain location. The information collected by both groups is stored in a spatial database and displayed on a Google Maps map. The application extends Google's Direction Service with obstacle avoidance functionality that enable users to find the shortest path in a disaster stricken area.

This project is carried out as part of the Crisis and Disaster Management course of MSc Geomatics and is supervised by Sisi Zlatanova.

ISBN: 978-90-77029-35-0
ISSN: 1569-0245

Section GIS technology
OTB Research Institute for the Built Environment
TU Delft
Jaffalaan 9, 2628 BX Delft, the Netherlands
Tel.: +31 (0)15 278 4548; Fax +31 (0)15-278 2745
Websites: http://www.otb.tudelft.nl
Http://www.gdmc.nl

E-mail: s.zlatanova@tudelft.nl

This publication is the result of the research programme Sustainable Urban Areas, carried out by Delft University of Technology

Contents

1 Introduction

Effective management of a disaster depends on knowledge about the health and condition of the infrastructure. The infrastructure is the basis for performing many spatial analyses. An important analysis in the response phase of a disaster management operation is the calculation of shortest routes between two locations while considering the damaged infrastructure. However, collecting information about the state of the infrastructure is a complex task due to the extent of the area and quantity of needed observations. Fortunately, making these observations does not require much specialization as almost everyone can judge whether a road is blocked. It therefore comes to mind to leverage the knowledge and large numbers of the crowd in collecting information about the infrastructure.

Crowdsourcing and web mapping are becoming increasingly more common in society as well as disaster management. Crowdsourcing has shown its strengths in endeavours such as Wikipedia. Web mapping platforms such as Google Maps and Bing Maps have revolutionized cartography and have brought it to the masses. New web technologies have made creating dynamic and intelligent websites easier. The combining of crowdsourcing and web mapping have produced OpenStreetMap (Haklay and Weber 2008). The field of Disaster Management has also benefited from this combination in the form of Ushahidi (Goldstein and Rotich 2008), a hazard mapping web application that used in disaster relief operations. Using Ushahidi the "crowd" is able to collect, store and share information about events and points of interest in the disaster area. They can identify blocked roads, shelter locations, people in need of immediate medical attention, etc. Rescue workers can use this information to quickly get an overview of the needed aid and plan their operations around that information. However, it does not support geospatial analyses. More specifically, it is not possible shortest route calculations.

This report presents our investigations on crowdsouring for disaster management and a small WebGIS application that enables the "crowd" to collect information that is used to calculate shortest routes between two locations. The focus of this report lies on the crowdsourcing aspect of disaster management. What is crowdsourcing? What constitutes crowdsourcing emergence? How should found constituents be implemented?

This report is organized as follows: section 2 discusses what crowdsourcing is and what stimulates its emergence and growth. Section 3 outlines the main idea while section 4 details the implementations. Section 5 presents conclusions while section 6 proposes next steps of research.

2 Crowdsourced disaster management

The sudden change of urban infrastructure configuration and health immediately after a disastrous event renders much of existing urban infrastructure information useless and out-dated. At the same time, information about infrastructure health is vital as a large part of rescue operations make use of roads, bridges and tunnels. In a disaster situation, finding the shortest route is a complicated operation that is no longer a function of distance and travel time only, but also of the infrastructure state. Obtaining up-to-date infrastructure information (quickly) is challenging due to many factors: the extent of the urban area, lack of specialists to investigate all the areas on the ground, drawbacks of remote sensing approaches for automatic damage detection, etc. Citizens and other non-specialists can greatly support collection of ground information if they are provided with appropriate tools.

Traditionally, urban damage is detected by deploying remote sensing techniques and platforms (Kerle et al. 2008, Zhang and Kerle 2008, Li and Chapman 2008). The platforms vary from airplanes to satellites. Although the coverage of remote sensing images is sufficient, certain difficulties prevent it from becoming an all-round solution. For instance, turning this data into useful information requires a considerable amount of time and effort. Furthermore, the collected data runs the risk of quickly becoming out-dated and it may not always be possible to collect it due to cloud coverage and bad weather. Lastly, some damages can only be recognised and assessed by ground observations.

Two things suggest outsourcing data collection to the crowd. First, the task of observing whether a road is damaged or not is not an overly complex task. It can be performed by almost anybody. Second, previous disasters have shown that a large group of people is willing and able to help. Goodchild (2007) recognizes the crowd's potential by highlighting that each individual is in essence a sensor, while the crowd as a whole forms a sensor network. Laituri (2008), Shirky (2009) and others show that the crowd is capable of more than data collection only. Examples of successful crowdsourcing disaster management operations can be found during the 2010 Haiti earthquake. Mappers from OpenStreetMap created a detailed map of Haiti in a matter of days while volunteers from Ushahidi helped translate a large number of SMS messages (Harvard Humanitarian Effort 2011).

2.1 Internet for ODRC

Disaster management is seen to be the task of official organizations such as governments and humanitarian non-governmental none-profit organizations. Until now, these institutions have largely remained off the internet. Laituri and Kodrich (2008) identify a move towards usage of internet technologies in the form of online disaster-response communities (ODRC). They identify three tiers. The first tier consists of a network of traditional national and international organisations that are responsible for raising awareness and financial funds prior to the occurrence of a disaster.

The second tier holds the parties and organizations that respond immediately after a disaster strikes. This tier is largely filled by organizations from the first tier. Instead of raising awareness, they are now coordinating action. Since recent years, the second tier is expanding to house informal organizations and networks that wish to contribute to disaster management and rescue operations. Their activities involve using internet and social media to share information about the disasters in the form of pictures, blogs, videos, wiki's and links to official sources of information.

The third tier is using the internet technology for more than information distribution only. This tier consists of technology savvy volunteers who are able to collect geographical information and/or build geospatial analysis tools that aid during, but also after the disaster management process. Examples of these endeavours are the OpenStreetMap mappers, Ushahidi creators and users, and communities such as Crisis Mappers[1] and CrisisCommons[2]. Third tier contributions are not limited to geographical information and "traditional" sensing and observation techniques. The 2011 Japan earthquake and nuclear power plant failure moved people to install Geiger counters and stream the measurements to the world through Pachube[3]. Botterell and Griss (2011) also identify this decentralization and "democratization" of disaster management activities and predict a move from the traditional "Command and Control" paradigm towards a social media powered "Cooperation and Coordination" approach.

2.2 Crowdsourcing

Crowdsourcing is the coming together of a diverse (in terms of knowledge, background, specialisation and interests) group of people who, using modern internet technology, perform complex tasks that are normally performed by specialists and professionals (Goodchild 2007, Goodchild 2010, Laituri and Kodrich 2008).

Crowdsourcing is the term used to denote the activities of the informal part of the second tier and the entire third tier. People's motivation to contribute to a project vary from altruism, to discontent with the speed and quality of traditional media (Sutton 2010), to hobby but also to mavenism (the urge to teach and educate others) and men's basic need to communicate and collaborate. The currency of volunteers is motivation and time instead of money, just like any other volunteering project (Goodchild 2007).

The best-known crowdsourcing product is Wikipedia, the online encyclopaedia. Advances in mobile devices and web mapping technologies combined with the crowdsourcing phenomenon have produced OpenStreetMap, a map that at some locations is more detailed and up-to-date than "official" maps. Yet, OpenStreetMap is more than an automatic aggregation of GPS tracks. Some parts are synthesized from satellite images by digitization. The "crowd" is thus capable of more than data collection only, it is able and willing to perform more complex task such as mapping, geographical analysis, translating SMS messages and developing small dedicated applications. Lately, crowdsourcing is slowly finding its way into disaster management (Lukaszczyk 2011).

[1] www.crisismappers.net
[2] www.crisiscommons.org
[3] www.pachube.com

Crowdsourcing has become possible due to recent advances in internet technology and specifically due to advances in communication technologies (Shirky 2009). Social media such as blogs, wiki's, Twitter and Facebook have increased ease of communication by providing more and better-streamlined communication channels. This decrease in communication transaction cost results in the emergence of highly dispersed but effective loosely organized groups of people who share a common interest.

Social media technologies allow groups to quickly organize and effortlessly manage themselves, thereby removing the need for a managerial layer that is inherent to large organizations (Shirky 2009). Such a group becomes cheaper, thereby allowing it to undertake tasks previously deemed too expensive in terms of time, money and manpower. The loose organization keeps the group flexible and agile, allowing it to quickly adapt to changing situations. The reduction in transaction cost enables "everyone to communicate with everyone" thereby exposing the product under development to many eyes and as many disciplines and expertises. The total amount of knowledge and expertise increases which in turn allows the completion of complex tasks (Shirky 2009, Laituri and Kodrich 2008). Amongst the numerous crowdsourcing virtues, the following are recognised to be valuable for disaster management.

Speed: crowdsourcing initiatives need little effort to materialize and start cooperating. An Ushahidi instance can be up and running in two hours. The OpenStreet mappers have produced completely new and highly detailed maps of Haiti in the course of days (Harvard Humanitarian Effort 2011). Traditional organizations tend to be slower in their response (Sutton 2010).

Up-to-date data: crowdsourced data can be collected at a tremendous pace and kept fresh due to the "many eyes watching" principle. A myriad of channels exist that can be used to monitor the relief operations from abroad. Information is shared easily through Ushahidi and Shahana, but also through blogs, Twitter, Facebook, etc. while geographical information can be distributed through platforms such as GeoNode and other OGC products.

Wide knowledge pool: as discussed in the previous section, crowdsourcing initiatives are characterized by a widely diverse group, both in terms of knowledge as location, of contributing volunteers. The “crowd” that gathers to help during an emergency is located near the occurrence (locals) as well as far from it (people assisting through the web) i.e. it is worldwide and hyperlocal at the same time. The advantages of this configuration are numerous. For instance, Heipke (2010) notes that giving local people power to contribute to crowdsourced data often results in higher quality of information than information that is gathered by someone not familiar with the surroundings. The reverse also holds true: a highly specialised and knowledgeable person can help even when he is located on the other side of the globe.

Momentum: Due to their openness (crowdsourcing initiatives use the web to communicate and open source tools to collaborate), crowdsourcing initiatives gain momentum faster and keep it going for longer than closed organizations. The openness of the systems allows new comers to gain speed quickly.

Continuity: a substantial part of volunteered (geographical) information or disaster management software is the product of free time activity and, to a lesser degree as a

by-product of commercial processes. As such, volunteers are constantly working on, and are surrounded by, the information and tools that they later deploy and use during a disaster management operation. The so created continuity ensures an efficient and effective deployment and usage of the technologies. Although official disaster management agencies organize training sessions, disaster management is often one of their many tasks and is certainly not a day-to-day experience.

The biggest threat to acceptance of crowdsourcing results, and especially data, has always been the question of reliability and robustness. Flanagin and Metzger (2008) discuss these issues in terms of *believability* or *credibility-as-perception*. The degree of *believability* is determined by trustworthiness and expertise. Flanagin and Metzger (2008), Goodchild (2007) and Shirky (2009) note that volunteered efforts can be trustworthy even when not produced by experts by relying on the collective "wisdom" of the crowd to detect and correct inaccurate information entries, keep the data set up-to-date and "defend" it from vandals and bugs.

3 Concept

The crowdsourcing virtues presented in section 2.2 suggest its deployment towards disaster management purposes. To this end, the classical Geographical Information Systems (GIS) main capabilities i.e. data storage, management, analysis and presentation are extended with the crowdsourcing mechanisms described earlier on in order to create a GI system fit for disaster management routing applications. Traditional GI systems are holistic, heavy weight solutions i.e. a single GI system is designed to solve a diverse set of spatial problems. GI systems need powerful desktop computers, constraining GIS experts to a desk. GIS 'in the field' e.g. in the hands of first responders and volunteers has not seen a lot of practical application. Recent advancements in mobile technologies have great potential to change the current situation.

The dynamic nature of modern web pages and applications made possible by Web 2.0 technologies has started to move GI Systems away from desktop machines. These technologies make it easy to connect mobile applications to GIS servers through the Internet in an interactive manner. Mobile devices thus become gateways to powerful servers that house geographical data and perform complex analyses. Such mobile and lightweight GI systems are called WebGIS. From the user's point of view, a WebGIS is capable of performing the standard GIS operations, but now users can take that functionality with them wherever they go. To test the applicability of these technologies and ideas, this report presents an application that brings crowdsourcing and GIS analysis to the mobile device.

3.1 Design choices and system architecture

The goal of proposed application is thus twofold: 1) automate the way finding process in a disaster stricken area by 2) enabling volunteers to act as sensors and report on the infrastructure health.

The application is aimed at the second and third tiers defined in section 2.1, namely users with limited familiarity with web technologies and computing principles in general, and computer savvy users who are able to work with the raw data and use it to build their own apps. Non-expert users interact with the application through a desktop interface and a mobile interface. Users indicate blocked roads by drawing polygons on a map. Desktop users asses the condition of the infrastructure by tapping into geographical information sources such as satellite images and official reports, but also into social media data sources such as blogs, forums, Facebook, Twitter, etc. The mobile interface allows for the collection of data through a mobile device such as a smartphone. Mobile users act as sensors and report on the status and health of the infrastructure. Expert users interact with the application by way of an Application Programming Interface. Figure 1 illustrates the system's configuration.

As already noted, the crowd is capable of more than data collection only. To leverage this capability, the built application allows for the calculation of the shortest route

between two points. The routing functionality turns the application into more than a data silo. The purpose of this analysis is twofold. On one hand it provides rescue workers with an automated shortest path analysis tool. On the other hand it acts as a return of investment for the crowd as they can use their own data for their own routing needs. The people's usage of their own data acts as an incentive to generate better and accurate data, and keep it up to date.

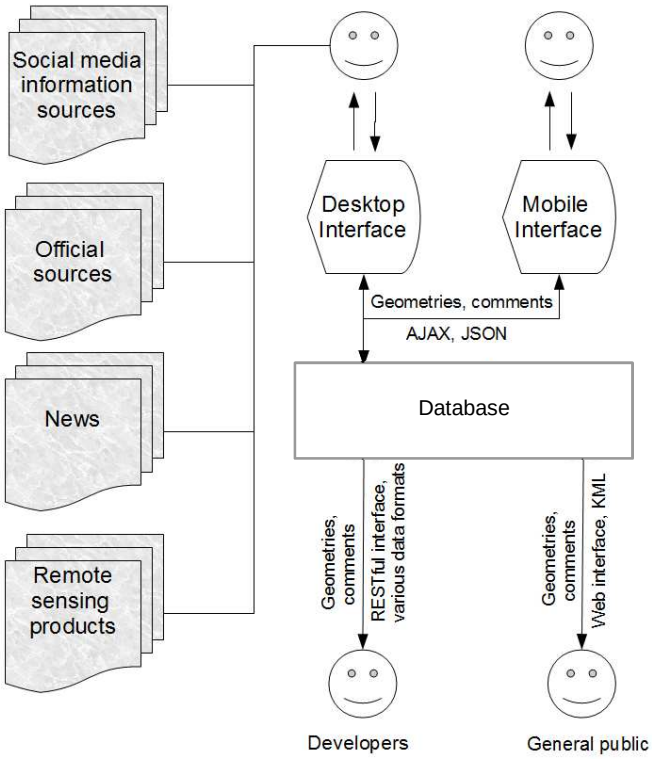

Figure 1 The system's components. Data streams are represented by arrows. The users on top are the data collectors. They gather data and enter it through the desktop and mobile interfaces. The text on top and left of the lines denote the payload of the data stream while the text on the bottom and right of the lines denotes the used technology.

Users are encouraged to cooperate on several different levels. First, they can work together on collecting the most accurate information they can find through all the means they are comfortable with. For instance, some may draw information from their Twitter network, whereas others may have some experience working with satellite images. Both types of users can enter their data in the application and compare the results. A third user may then check both of their results. Another type of cooperation is found between the desktop and mobile users. As the mobile user moves through the disaster area he is makes quick and numerous observations and uploads them to the system. The desktop user then synthesizes the observations into polygons. Both users benefit: the mobile user can work autonomously without worrying about what other mobile users are doing, while the desktop user can observe the whole operation from a higher vantage point using more powerful hardware without needing to worry about the difficulties of working in the field.

Crowdsourcing is the formation of a group of people, a network. The formation, activation and maintenance of this network is achieved through communication. Here, communication is understood to be broader than the bare minimum needed to

achieve a certain degree of collaboration. Tornqvist et al (2009) explain that casual communication, communication outside the immediate scope of the work at hand, is the instrument of trust building among cooperating parties. Hence, crowdsourcing thrives only when numerous and diverse communications channels are available. Every platform willing to support crowdsourcing efforts must therefore provide means for the participants to communicate. On top of that, the geographical nature of disasters calls for an easy mechanism to share and view spatial data. Following is a list of requirements based on the previous two sections deemed necessary for the emergence of crowdsourcing for disaster management and solutions that are implemented in the here presented application.

- Communication is facilitated by providing users with means to discuss all aspects of their efforts through a myriad of communication techniques such as messages, chat, tags, etc. Linking to already existing sources of social information can be done by embedding social media like a Twitter stream (with a designated hashtag) in the application/page the crowd is working on (see Figure 2), but also links to discussion groups, Facebook pages and others as they become relevant and important. The general idea is to connect the application to as much already existing ways of disaster management communication channels as possible.
- Disaster management information is a diverse as there are platforms to disseminate it. Tapping into as many information streams as possible is vital for the successful generation of a situational awareness and overview. Data sharing is achieved by providing access to the data by supporting different file formats, databases and webservices as is explained in section 3.2
- Quality, reliability and correctness of data can be guaranteed by building a large community which, as explained in section 2.2, sets out to 'guard' the data by performing cross-checks, removing false entries and assessing the credibility of users and data sources i.e. some degree of trust must be created and maintained. This is stimulated by enabling communication amongst users and keeping track of volunteers who produce quality information and rewarding them as suggested by Heipke (2010) and Flanagin and Metzger (2008). User trustworthiness can be verified by checking their online track record. In the case of Twitter, indicators for authenticity and trustworthiness are the number of tweets, number of (credible) followers, number of retweets, etc. (Meier 2011).

Figure 2 Fukushima radioactivity map. A good example of a crowdsourced ad-hoc project. Volunteers set out to place Geiger counters in the field. The measurements are streamed through Pachube to the net. The left panel shows a Twitter stream displaying all tweets with the #fukushima tag.

Although several web based emergency management systems are available, they often lack features and analyses that are needed in a crisis. Acuna et al (2010) have compiled a list of design patterns that any given disaster management application should incorporate. A subset of their findings is combined with the crowdsourcing needs discussed in the previous section and the routing functionality discussed earlier to form a set of information presentation functionalities that are deemed relevant for implementation viz.

1. *Awareness for first responders: fast and dynamic access to information regarding the emergency at hand.* This design pattern is split in two: the speed with which the information gathering system can be deployed, and the speed with which information can be extracted from it. The developed application targets both aspects by realizing a quick system set-up by relying on off-the-shelf technologies such as mobile phones and providing an easy to use interface to the information.
2. *Map-based navigation and information presentation*
3. *Tabular information presentation*
4. *Data authoring: mechanisms for attaching author and source information to data items*
5. *Mechanisms for direct data manipulation*
6. *Display of up-to-date data*
7. *(Temporal) data archives*
8. *Support for hand-held devices*

The remainder of this report discusses how the functionalities are implemented alongside the routing capabilities.

3.2 Technology

The design choices and system architecture presented in the previous chapter require a number of technologies in order to operate and reach the set goals. This section gives an overview of the chosen technologies alongside short explanations about each.

The built application is a client-server configuration. The client side runs on HTML/JavaScript and communicates through *asyncrhonous Javascript and XML* or AJAX (see Appendix A: Saving geometry to PostGIS for code examples) with a RESTful server. The JavaScript library jQuery[4] is used to implement AJAX. REST is an abbreviation of REpresentational State Transfer, a "software architecture for distributed hypermedia systems such as the World Wide Web" (Wikipedia). In this architecture, every resource is stored on a server and has a unique identifier. In web-based systems, this identifier is known as a Unified Resource Identifier (URI). Clients can access and manipulate the resources by standard HTTP methods. Obtaining the current state of a resource, for instance, is done by sending a HTTP GET request to a resource's URI. The server processes the request and sends a representation of the resource's current state as a XML, JSON (the lightweight data-interchange format JavaScript Object Notation), plain text, etc. document. The interface is built using standard HTML and the JavaScript library jQuery UI[5]. The used mapping framework is Google Maps[6].

Here, REST is chosen instead of the more traditional web service protocol SOAP, since REST is deemed lighter and easier to deploy. REST is based on and uses well-known, proven and implemented W3C/IETS standards (HTTP, URI, XML, etc.). Deploying RESTful web services is therefore relatively simple as the needed infrastructure already exists. Also, resource state representations can be sent using lightweight data formats such as JSON (Pautasso et al 2008).

SOAP, on the other hand, defines an XML protocol for exchanging structured information. SOAP is more extensive as it provides specifications and means for dealing with transactions, security, reliability, protocol transparency, etc. SOAP is geared towards "enterprise" environments. This extensiveness results in a relatively complex SOAP stack. The built application relies on simple “get” and "set" operations and does not require the functionality provided by SOAP. SOAP requires custom client-side software to run tests (Pautasso et al 2008). REST, on the other hand, can be tested through a web browser by pointing the browser to the resource’s URI that is in essence an URL in web-based systems.

REST is implemented using Django[7]. Django is a Python web application framework that eases web application development by automating and abstracting low-level tasks and operations and by automatically building the database schema. Developers can then focus on the application logic. The GeoDjango[8] plugin makes Django spatially enabled. GeoDjango uses open source libraries such as GDAL/OGR and GEOS to interact with and manipulate geographical information. Django, and by extension GeoDjango is database agnostic. Since all is written in Python, any

[4] http://www.jquery.com
[5] http://www.jqueryui.com
[6] http://maps.google.com
[7] https://www.djangoproject.com/
[8] http://geodjango.org/

geospatial library with a Python API can be used to perform spatial analyses. GeoDjango, as REST, has been chosen for its simplicity, ease of deployment and very high quality documentation. Alternatives such as GeoServer are full-fledged applications that offer functionality than is currently required.

PostGIS[9] is chosen as the main data store as it is a well-known, powerful, open source spatial database. It enjoys wide usage amongst open source developers. The developer community is able to set it up quickly and perform advanced tasks. PostGIS enjoys the momentum discussed in section 2.2.

In the current set-up, it is the 'expert' (i.e. developers) entry to the data. Developers access the data through a RESTful Application Programming Interface (API). They can build applications on top of the collected data. The PostGIS and Django/Python combination is powerful as one can use PostGIS' built-in spatial functions, as well as spatial libraries and software packages that are accessible with Python e.g. GRASS, ArcGIS, GEOS, Shapely, GDAL/OGR, etc.

Google Fusion Tables[10] is "a cloud-based data management and integration service" that is designed specifically with collaboration, data sharing, the Web and usage by non-technical users in mind (Gonzalez 2010b). Fusion Tables stores (geographical) data in tabular form. Collaboration is supported by allowing users to comment on and share data. Users are able to comment on tables, columns, rows and cells. Tables can be shared with the world or a selected group of people all of which can have different roles i.e. viewers (view and comment) and editors (view, comment and edit). The data is accessible through a Web interface and can be visualized in several different ways, the most interesting being as geometry on a map. Tables storing geographical information can be exported as KML. Google Fusion Tables is thus the preferred way of data access for non-expert users. Note that users cannot add information through Fusion Tables.

The used mapping framework is Google Maps[11] (GM). GM is chosen as the underlying platform since it is robust and accessible by all on a diverse number of devices. Google Maps is hosted in the cloud which guarantees its availability and makes it fast since all involved calculations are computed on powerful servers. An internet connection is all that is needed for the proposed application to work. Second tier or non-expert users are visually accustomed to it and know what to expect. The well-documented API has made Google Maps popular amongst third tier developers.

The WebGIS application is hosted on the author's website[12]. The source code is hosted on the code collaboration website Github[13].

[9] http://postgis.refractions.net
[10] www.google.com/fusiontables
[11] http://maps.google.com
[12] http://gmer.ndkv.nl/
[13] http://www.github.com/ndkv/gmer/

4 Implementation

Based on the crowdsourcing and disaster management design patterns discussed in section 3, a small WebGIS is implemented that aims to enhance the disaster management activity of route finding with information gathered through crowdsourcing.

The developed application sticks to the standard GIS functionalities of data gathering, storage, analysis and visualisation, but extends these with crowdsourcing and disaster management mechanisms.

4.1 Data gathering, storage and management

The developed application has two interfaces that facilitate information input. The desktop interface is accessed through a desktop/laptop internet browser and is designed for volunteers that are away from the disaster area. They input obstacle information by drawing polygons on a map that denote blocked infrastructure areas. This information can come from anywhere: news reports, satellite images, Twitter, etc. Desktop users also check, validate and clean the mobile users' input (Figure 3).

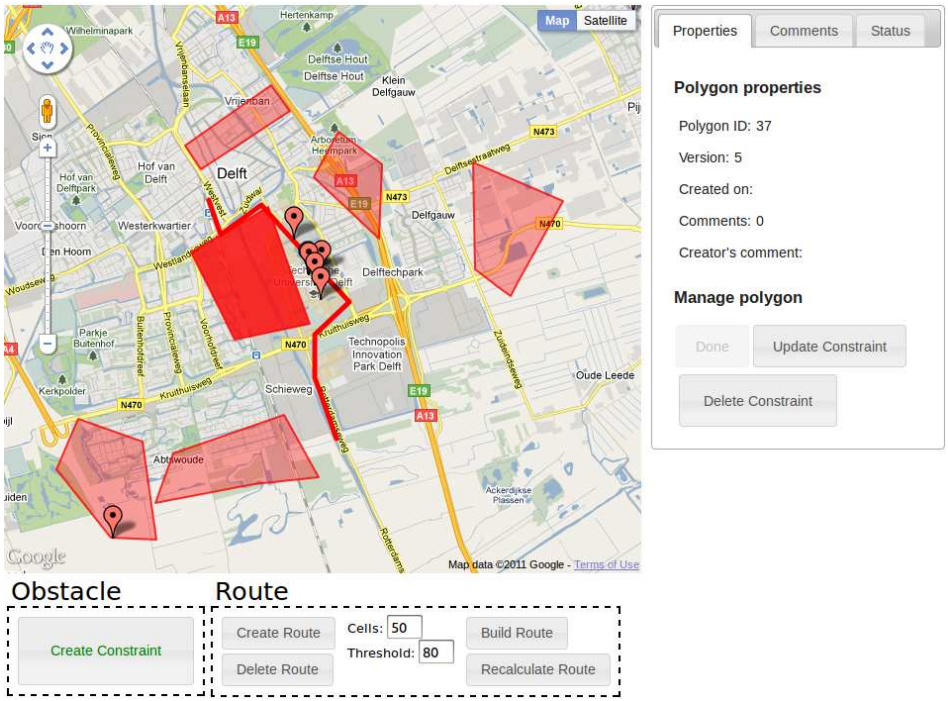

Figure 3 The desktop interface. Polygons denote real-world obstacles.

TODO: explain the interface more thoroughly.

The desktop interface has four panels: a map, obstacle and route creation controls, obstacle properties panel and a comments panel. Users create obstacles and routes using the controls under the map. The entered obstacles are versioned and stored indefinitely in the database. Each obstacle therefore has a history. Temporal data is useful during a post-mortem where it sheds light onto the development of the

disaster as well as the behaviour of the people on the ground. Figure 4 and Figure 5 show the *versions* table. Each version refers to an obstacle through a foreign key relation on the obstacle_id field.

Column	Type	Not Null	Default	Constraints
id	integer	NOT NULL	nextval('gmer_version_id_seq'::regclass)	
obstacle_id	integer	NOT NULL		
version	integer	NOT NULL		
date	timestamp with time zone	NOT NULL		
geom	geometry	NOT NULL		

Figure 4 A snippet of the table storing the obstacle versions.

Routes are defined by providing a point of departure and a point of arrival after which the implemented routing algorithm together (discussed in section 4.2) with Google's Directions Service calculates the shortest path around all polygons.

id	obstacle_id	version	date	
12	15	0	2011-12-21 13:59:37.018307-06	POLYGON((4.30975287780757
13	16	0	2011-12-27 00:29:57.988955-06	POLYGON((4.4517168365478 5
14	17	0	2011-12-27 00:41:57.059508-06	POLYGON((4.37137931213374
15	18	0	2011-12-27 09:26:28.984052-06	POLYGON((4.33850615844722
17	20	0	2011-12-29 05:45:32.147215-06	POLYGON((4.4224485717773 5
18	21	0	2011-12-30 09:05:59.048978-06	POLYGON((4.37163680419917
19	22	0	2012-01-03 04:05:41.782862-06	POLYGON((4.49832290039058
20	23	0	2012-01-16 10:52:46.227501-06	POLYGON((4.40202086791987
21	24	0	2012-01-16 10:54:04.456997-06	POLYGON((4.34708922729487
22	24	1	2001-12-01 00:00:00-06	POLYGON((0 0,10 0,5 10,0 0))

Figure 5 Database dump of the versions table.

The panels to the right of the map harbour the crowdsourcing mechanisms discussed in section 2.2: the *properties* panel provides an overview of obstacle metadata (e.g. id, owner, creation date, type, etc.), the controls under *manage polygon* allow one to modify and delete obstacles, while the *comments* tab allows users to discuss obstacles by leaving comments.

Comments are kept indefinitely and displayed one under the other thereby creating a log that holds information about the evolution of each obstacle, as well as the discussion belonging to it. This information is useful during the disaster as it allows new contributors to quickly get a sense of the situation's dynamics. It is also useful during the debriefing phase as it allows one to reconstruct an accurate picture of the events. Figure 6 displays the comments PostGIS table.

Column	Type	Not Null	Default	Constraints
id	integer	NOT NULL	nextval('gmii_comment_id_seq'::regclass)	
obstacle_id	integer	NOT NULL		
content	text	NOT NULL		
date	timestamp with time zone	NOT NULL		
user	character varying(100)	NOT NULL		

Figure 6 The comments table.

id	obstacle_id	content	date	user
6	23	fsdsdfsd	2012-01-16 10:52:54.433416-06	Klaas
7	23	fsdsdfsd	2012-01-16 10:52:54.841296-06	Klaas
8	23	jhjghj	2012-01-16 10:53:04.617109-06	Klaas

Figure 7 A dump of the comments table

The mobile interface is a simplified version of the desktop interface. It runs on any modern smartphone that has an internet connection and a browser that is able to run Google Maps (Figure 8). The mobile interface serves two purposes. On one hand, it allows users to walk around the disaster stricken area and perform observations about the state of the infrastructure, while on the other it acts as a real-time information source for rescue workers. The two interfaces are linked i.e. obstacles that are added via the desktop interface are visible on the mobile one and vice versa.

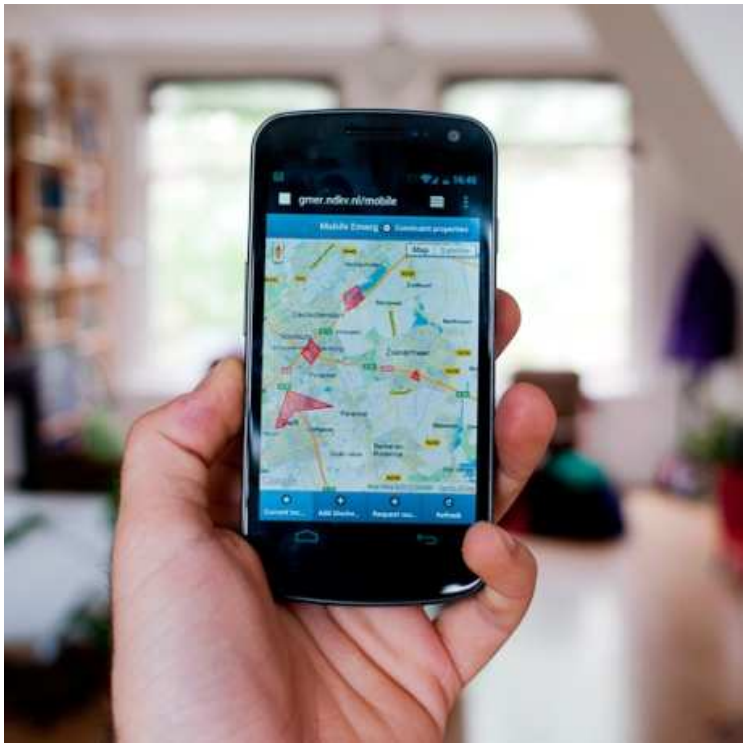

Figure 8 The mobile interface.

The application and its two interfaces are open to all i.e. users do not need an account to add, edit and remove information. This is meant to lower the barrier to participation. Allowing users to quickly edit an obstacle guarantees the data's up-to-dateness, encourages volunteers to actively participate in the process and allows the crowd's self-correction principles (as explained in section 2.2) to kick in.

The collected information is stored in a PostGIS database. For each obstacle, the following information is stored: obstacle ID, creation date and creation comment. This can easily be extended to store other properties such as observer role: volunteer / specialist/ journalist, verification status: verified / under investigation, obstacle type: rubble / water / holes, obstacle source: observed / mapped, possible passage: walking / driving / trucking / etc.

At the moment GFT stores the polygon's geometry only. Data is stored in PostGIS and Fusion Tables immediately after a user creates a polygon i.e in real-time. An

AJAX GET request sends the data to the server where it is stored in PostGIS through Django while a Python API is used to write the data to Google Fusion Tables. See Appendix A: Saving geometry to PostGIS for code listings.

4.2 Analysis

The routing algorithm mentioned above is the heart of the application's analysis. The shortest route calculation is summarized as follows: an initial Google Maps Directions Service result is generated from the user defined start and destination points. The Directions Service (DS) runs on Google's servers and calculates the shortest route between the provided start and end points. This result is intersected with the obstacle polygons. Each polygon is avoided by calculating a path around it by way of an A* pathfinding algorithm that uses the intersection points as start and end points. In the current implementation, the intersections are found by deploying simple computational geometry algorithms. The A* result is simplified and used as waypoints for the Directions Service. Some manual adjustments of the returned route are necessary.

Ideally, shortest path calculations are graph based. However, Google Maps does not expose its vector data e.g. it is not possible to extract a graph of the streets. Obstacle avoidance and shortest path analyses are therefore performed in the raster domain.

Google Maps does not provide mechanisms to check whether a point is contained by a polygon. Point-in-polygon analyses are used for line intersection detection and rasterization of polygons. Checking if a point is contained by a polygon is done using the winding number algorithm (Worboys 1995). The winding numbers algorithm calculates and sums the angles between a point and all polygon edges. If the summation equals $2\ \pi$ then the point is said to be in the polygon.

Shortest path analyses are performed using the A* pathfinding algorithm (Hart et al 1968). Lines are simplified with the Douglas-Peucker algorithm (Douglas and Peucker 1973).

4.2.1 Routing algorithm implementation

The shortest route finding process can be split in the following major parts.

1. Obstacle, path and initial route calculation: the user defines the obstacles as well as the start and end points of the route. An initial route is calculated by Google's Directions Service.
2. Route and constraint intersection: intersections between the initial route and obstacles are found using the algorithms described in the previous section.
3. Shortest path analysis and visualization: the intersections found in step 2 are passed to the A* pathfinding algorithm which finds the shortest path around the obstacle. A new shortest route is requested from Google's Directions Service which is obliged to pass through the points calculated by A*.
4. Result adjustment: the result from step 3 is not perfect and needs minor manual adjustments.

Step 1 Obstacle, path and initial route definition Obstacles are defined by drawing a polygon on the map in a clockwise order. Markers are displayed to identify the polygon vertices. The obstacle creation process is ended by a right mouse click. Next, the user needs to enter the route start and end points. A left click identifies the start point while a right click identifies the end point. The shortest route calculation is performed by the Google Maps' Directions Service.

The Google Maps Directions service takes a begin and end point and calculates the shortest route connecting both points. The Directions Service returns turn-by-turn driving directions. Each turn instruction has a latitude and longitude coordinate. However, the driving instructions' main purpose is navigation. As such, a turn instruction is given only when a turn actually has to be made. It is therefore impossible to predict the number and locations of received latitude/longitude pairs. The route returned from the Directions Service is therefore defined by a list of randomly placed coordinates. For example, long stretches of road will be represented by two coordinates only: one belonging to the instruction stating to get on the road and another to the instruction stating to get off the road, since the driving instruction is of the form 'Turn left on Rotterdamseweg'.
This behaviour makes finding intersections between the route and obstacles difficult as it creates a number of intersection scenarios which need to be treated separately.

Step 2 Route and constraint intersection The aim of the intersection detection procedure is to find the two vertices which lie just outside the obstacle polygon. These will function as start and end point for the A* pathfinding algorithm.

Figure 9 identifies the different intersection possibilities between the route and the obstacle polygon. The polygon is represented by the red area. Its bounding box is also given in the figure. The route segment coordinates returned by the Directions Service are represented by the diamonds and white dots in the polygon.

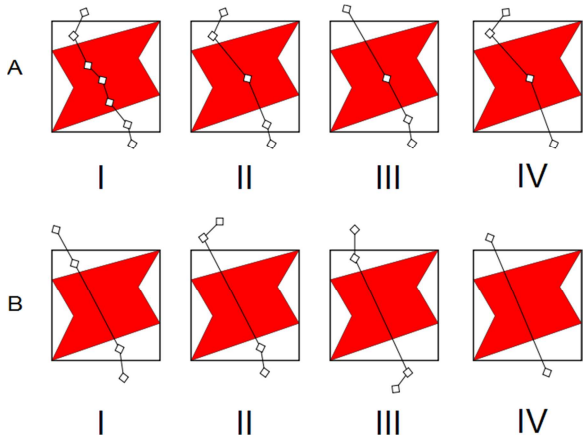

Figure 9 Intersection modes between the obstacle polygon and route returned from the Directions Service

Category A is characterized by the presence of a route vertex in the polygon. Category B is characterized by the presence of one or more vertices in the polygon's bounding box but none in the polygon. Case B.IV is special as no vertices are present in the bounding box but the segment does intersect the polygon. Two different intersection techniques are used for both cases.

Category A: The algorithm starts by checking whether any of the returned route vertices (the diamonds in Figure 9) lie within the bounding box of the obstacle. This is determined using Google Maps' built-in LatLngBounds object's contains() function. For all vertices which intersect the bounding box a point-in-polygon test is performed. Cases A.I-A.III are handled in the same way. First, the first vertex which lies inside the polygon is found. The previous vertex is then set to be the A* starting point. The end point is set to be the first point which is not contained by the polygon. Case A.IV is a a variation on the previous cases since the end point lies outside the bounding box.

Category B: A different approach is needed for category B since no vertices lie inside the polygon but an intersection exists. A point-in-polygon test will not work. Therefore, a line intersection algorithm has been implemented which intersects all route edges with all polygon edges. The line intersection algorithm is based on the side test mentioned previously.
In the current implementation cases I-IV are handled in the same way. To optimize the algorithm, only route segments having vertices contained by the bounding box are intersected with the polygon.

Step 3 Shortest path analysis As mentioned above obstacle avoidance shortest path analyses are performed in the raster domain. Obstacles provided by the user are rasterized. This is done by creating a grid around the polygon and checking which grid cells are contained by the polygon using the aforementioned winding numbers algorithm. Once the intersection points are found and the obstacle has been rasterized, the A* pathfinding algorithm is used to calculate a path around the obstacle. A result of the A* shortest path algorithm is shown in Figure 10 (in red).

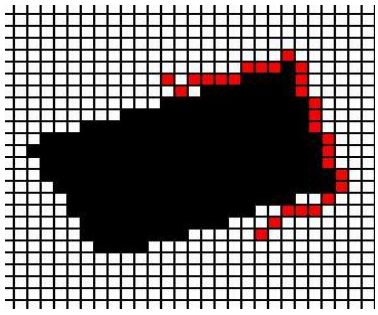

Figure 10 Result of the A* shortest path algorithm

The A* algorithm returns too many nodes. These are not needed and are done away with using the Douglas-Peucker (DP) simplification algorithm. The sensitivity of the DP algorithm is controlled by a threshold: points which are not significant for the shape of the line are removed. The DP result is used as waypoints for the second Google Maps Directions Service call.

Step 4 Visualisation and result adjustment The A* algorithm has no knowledge about the road network. The returned results will be far from perfect. A certain amount of modification will always be necessary. To facilitate this, the DP result is plotted alongside the Directions Service result. Making adjustments to the initial result is done by dragging the DP vertices to appropriate locations (see next section for more details). It is also possible to vary the size of the grid and the DP simplification threshold. Together, these variables control the spacing and amount of waypoints. A larger value for the DP threshold results in less waypoints as only

points which are far away from the line connecting the begin and end point. After all modifications have been performed, the user can invoke the Directions Service again to get a new shortest route.

4.2.2 Routing tests and results

Two examples are discussed in this section. The first example shows the result of a routing request in Delft containing two obstacles. The second example shows the result of a routing request near the bridges of Rotterdam. Figure 11 shows the cleaned result of the first example. The DP result is represented by the straight line segments marked by the white dots circles. In this case these are six. A Directions Service waypoint is located at every DP vertex. The Directions Service result is the markerless route which snakes through the streets and avoids the two obstacle polygons. The obstacle polygons are represented by the light red areas.

Figure 12 shows the result presented in Figure 11 prior to the quick manual adjustment. This initial result is, as explained before, not perfect. A basic understanding of the workings of the system is needed in order to correctly/optimally define the obstacles and improve the initial result (shown below) and obtain a cleaned route.

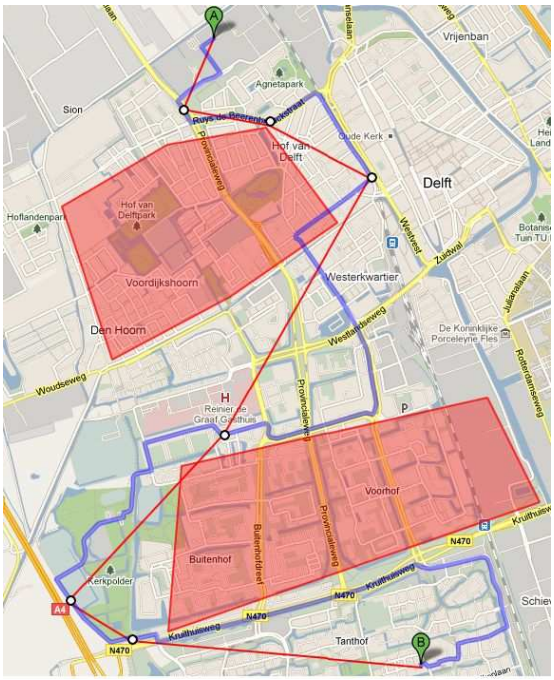

Figure 11 Cleaned result

When defining obstacles, one should keep in mind that the A* algorithm has no knowledge of the road network and that touches the obstacles when it avoids them. Obstacles have to therefore be extended to touch (but not cover) roads which are accessible for travel. The second example illustrates this issue (Figure 13). The obstacle defined on top of the two bridges is not extended far enough to the left. Since it is not touching the bridge on the left, the DP solution passes over the water to the left of the obstacle. Although the DP result successfully avoids the obstacle, the Directions Service is unable to calculate a path through the supplied waypoints (identified by the markers) as these are located over water. Figure 14 shows the correct obstacle definition i.e. the obstacle is touching the bridge on the left.

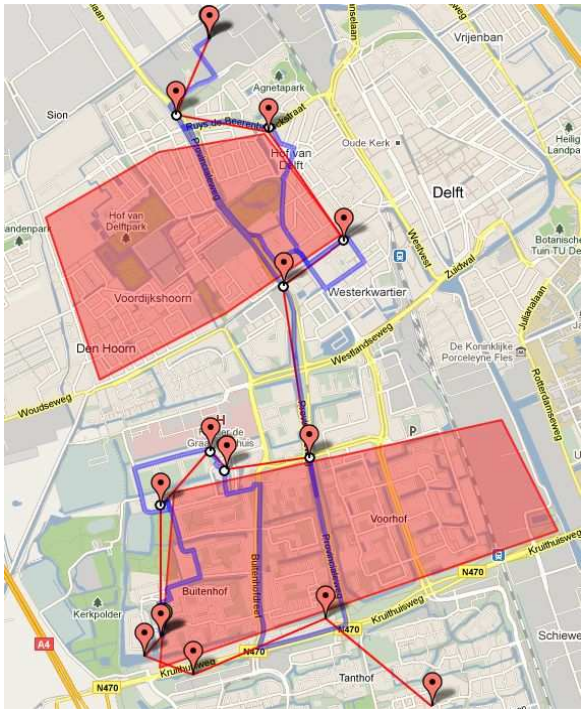

Figure 12 'Raw' result of routing calculations.

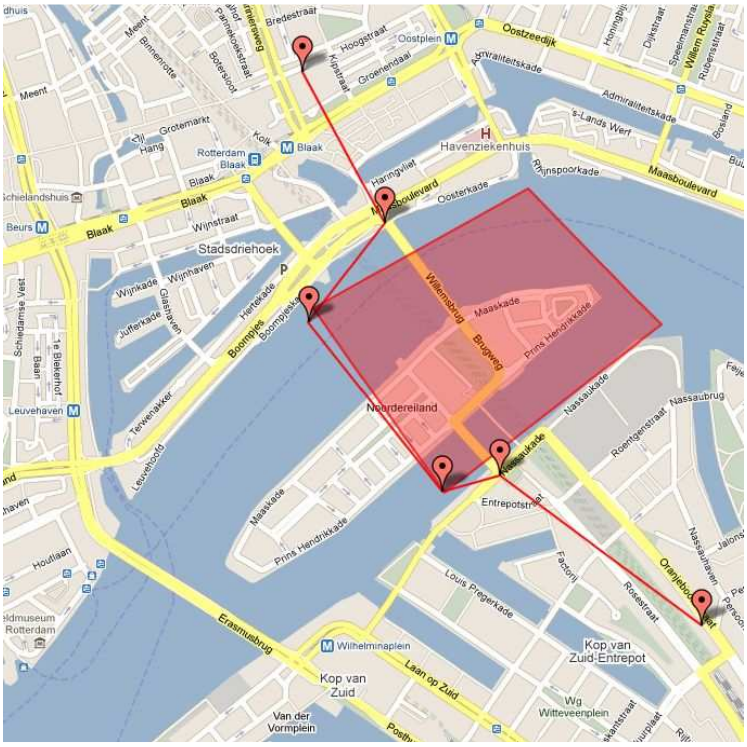

Figure 13 Demonstration of an improperly defined obstacle

Figure 14 Demonstration of a properly defined obstacle

The mentioned lack of access to vector data also influences the shortest route result in several other ways. For instance, some DP waypoints may simply fall on the wrong road. This results in a spaghetti like route as shown in Fig. 4. Another issue arises due to Google Maps' automatic snapping of waypoints to streets. While being and advantageous feature (the application would not work otherwise) it tends to cause problems when the DP result snaps to a small one way road instead of the neighbouring high way. Further, Google Maps is aware of street directions and the Directions Service obeys these. Waypoints which happen to be on the wrong side of the road cause the Directions Service to drive twice over the road in order to pass over the given waypoint. Lastly, if the DP threshold is too low i.e. the A* result is not simplified, a lot of waypoints will be returned to the Directions Service. This is troublesome in cities with small streets as the route will be made to go through a large number of them.

4.3 Data presentation, visualisation and sharing

The last segment in the GIS chain is the visualisation of information and analysis results. The built application has several visualisation channels that caters to the two user groups targeted here: "normal" users (informal part of second tier) who are not able or willing to work with raw data and prefer a pre-processed version of it, and savvy computer users (volunteers in the third tier) who want access to the raw data.

The first visualisation channels is the information is visualised on the map as displayed by the desktop and mobile interfaces. There, users are able, next to inputting data, to browse and interact with the already collected information. Google Fusion Tables (GFT) forms the second visualisation method (Figure 15). GFT's web-interface supports tabular and mapped visualisations thereby allowing the (amateur) users to explore the data in a familiar environment.

As discussed in section 2.2, crowdsourcing initiatives are successful and gain momentum when they are open and the data they handle is easy to share. Open here means more than just 'accessible'. The collected data must also be usable i.e. it must

also be structured, provided in a myriad of formats, be easy to browse and query and be easy to share amongst users.

The built application satisfies the two user groups' sharing needs through the following means: the web and mobile interfaces presented in section 4.1, GFT's export functionalities and visualisations, and the developer REST API. GFT allows data to be exported as KML. Since GFT is hosted in the cloud, sharing information is as easy as distributing a link to the data. The REST API gives access to the data in developer friendly formats such as Well-known Text, KML, GeoJSON, etc. and allows them to use the data in their own applications.

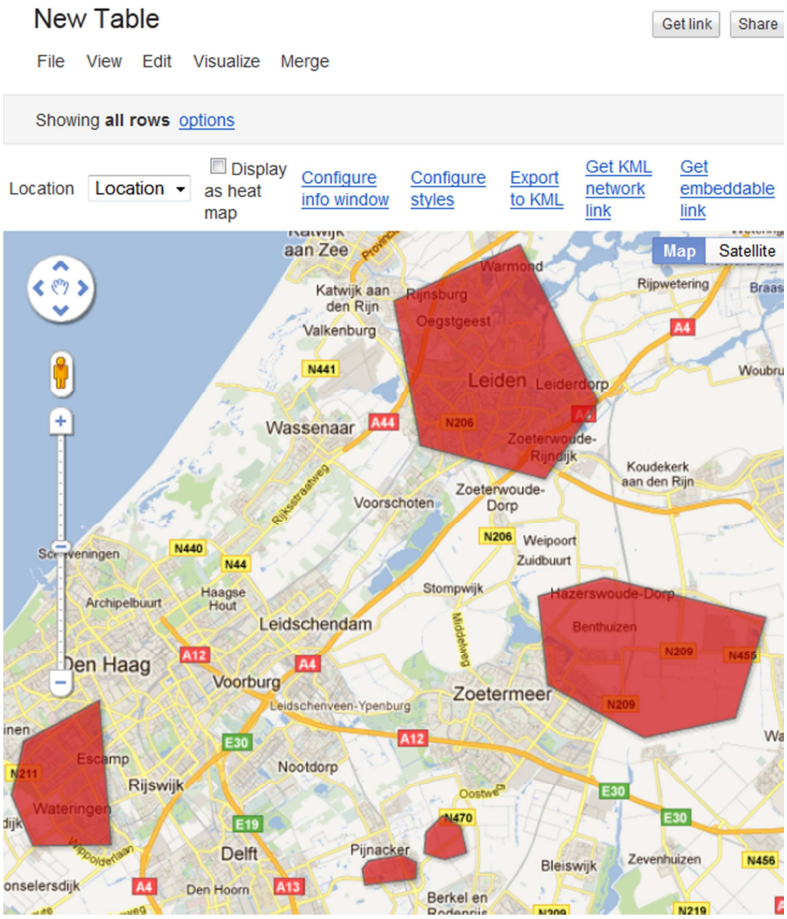

Figure 15 Google Fusion Tables' map view. Users can export the data to KML.

5 Conclusion

This report presents the implementation of a small WebGIS application geared towards the enablement and usage of crowdsourcing efforts for the collection of urban infrastructure health information. Based on the collected information, the application is able to calculate the shortest route between two points whilst taking the volunteered data into account. The application is built on top of PostGIS and Google Maps using AJAX techniques. A RESTful server is implemented using the Python web framework Django. Google Maps has been chosen for its speed, robustness and widespread use amongst amateur as well as more experienced computer users. PostGIS is the de facto geospatial open source database of choice. Django speeds up the development iterations as it takes care of low-level operations and tasks such as database creation and communication. This combination of well-known, high level tools allows the quick adaptation of the application to ever changing disaster management circumstances.

Crowdsourcing is the unplanned coming together of a highly diverse group of people who use the latest technology and data formats and sources available to aid a certain disaster management cause. It is difficult to predict beforehand how many people with what skills will participate and which tools they will deploy. Designing an application for a crowdsourcing effort therefore seems counter to its volatile nature. What is needed, rather, is not a complete system, but a set of components which are well developed, well documented and can seamlessly be integrated on the fly to provide to the situation's needs.

The main strength of the built WebGIS compared to other web disaster management solutions is its ability to perform spatial analysis in the form of routing. The application users are not only gathering data for others e.g. relief organizations, but also for themselves as they too can use the routing service. The quality of provided information is expected to rise once data gatherers experience first-hand how the provided data is used and how errors affect the routing solution. Data providers and gatherers' mindset might change from "contribute occasionally and forget" to "contribute continuously and guard quality". The implemented communication methods aim at creating a long lasting community.

Parallels between upcoming crowdsourced disaster management initiatives and the open source communities are important to notice and foster: computer savvy users fiddle daily with the technology they later use for disaster management. As such, they have momentum and a running, hands-on experience with used technologies. Using open source is vital as it enables hackers to adapt the software to their needs on the fly. Open sourcing a project allows more people to get involved, which in turn results in a larger knowledge and contributing user base.

The routing system works best in complex regions. Complexity in this case means a large number of streets and/or a large area of operations and/or many spread out obstacles. In these cases it becomes impossible to manually define a route which is optimal in a sense. A* guarantees that its result is the shortest possible path around the obstacle. The Directions Service also finds the shortest route. The obtained route

is the result of the stacking of two optimizers. Such a high degree of optimization is difficult to achieve by an user who is manually searching for the shortest path by maps observation only.

The routing system is especially useful when used by people who are not familiar with the layout of the city and the different types of roads. What might look shorter on a map need not be so in reality since, for instance, a shorter route may be slower in terms of time due to a lower speed limit or limited vehicle capacity.

6 Future work

The application's fitness for use has to be evaluated by deploying it in a real-world simulation. The implemented ideas and principles are based on theory only.

Trust is an important commodity in crowdsourced projects. Data source trustworthiness is, for better or for worse, often the main and only indicator of volunteered data quality. Mechanisms for increasing trust and checking the trustworthiness of data sources should be researched and implemented. Several ways of trust generation have been brought forward, one of which is communication. By communicating with volunteers, one is able to make a better estimate of their truth worthiness. For the current application, an extra communication channel is, for instance, application-wide chat . Users will then be able to discuss all aspects of the application, not only the obstacles. By implementing a user management system through, for instance, Twitter further strengthens the trust validation process by allowing to examine the contributors' history through their past Twitter activity. An user management system also allows the rewarding of quality contributors.

An in-depth study of obstacle input methods has to be performed. Maps are cultural entities and are therefore perceived differently by different communities. Drawing polygons may not be the most intuitive input method available. Research and field trials may be needed to assess the best input methods. It is suggested to evaluate whether inputting points is easier at both ends of the application: user's generating data and users validating data.

Currently, an internet connection is required for the prototype to function. The availability of working wireless networks cannot be taken for granted during disasters. Therefore a caching mechanism needs to be implemented which enable the prototype to function in the absence of internet connectivity. People will be able to go out, observe, save these locally on their devices and when near a network sync their devices with the servers. Alternatively, use can be made of SMS and GeoSMS (Chen and Reed 2012) to transmit information as the networks they rely on have shown to have a higher rate of survivability and/or are easier to set up.

Google Maps has been chosen as the mapping platforms due to its (Directions Service) speed, fitness for mobile devices, extensive adoption and documentation, but also due to a lack of open source solutions at the time of the here presented application's inception. The situation has changed considerably. Open Source tools and initiatives have taken a flight. The OpenStreetMap mapping success in Haiti suggest using OpenStreetMap data in combination with OpenLayers.

Bibliography

Acuna, P., Diaz, P., & Aedo, I. (2010). Development of a design patterns catalog for web-based emergency management systems. In *Proceedings of the 7th International ISCRAM Conference*. ISCRAM

Botterell, A., & Griss, M. (2011). Towards the next generation of emergency operation systems. In *Proceedings of the 8th International ISCRAM Conference*. ISCRAM.

Douglas, D.H., & Peucker, T.K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *10*(2), 112–122.

Flanagin, A., & Metzger, M. (2008). The credibility of volunteered geographic information. *GeoJournal*, *72*(3), 137–148.

Francia, S. (2011). Soap vs. rest.
URL http://spf13.com/post/soap-vs-rest

Goldstein, J. and Rotich, J. (2008). Digitally Networked Technology in Kenya's 2007 – 2008 Post-Election Crisis. IN *Berkman Center for Internet and Society: Internet and Democary Case Study Series.*

Gonzalez, H., Halevy, A.Y., Jensen, C.S., Langen, A., Madhavan, J., Shapley, R., Shen, W., & Kidon, J.G. (2010b). Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, (pp. 1061–1066). New York, NY, USA: ACM.

Goodchild, M. (2007). Citizens as sensors: the world of volunteered geography. *GeoJournal*, *69*(4), 211–221.

Goodchild, M.F., & Glennon, J.A. (2010). Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth*, *3*(3), 231–241.

Haklay, M. and Weber, P. (2008). Openstreetmap: User-generated steetmaps. *Pervasive Computing 7(4), 12-18*

Harvard Humanitarian Effort (2011). Disaster relief 2.0 report: The future of information sharing in humanitarian emergencies. Washington, DC and Berkshire, UK.
URL
http://issuu.com/unfoundation/docs/disaster_relief20_report?AID=10829131\&\#38;PID=4165814\&\#38;SID=1wg2zzyx464sp

Heipke, C. (2010). Crowdsourcing geospatial data. *ISPRS Journal of Photogrammetry and Remote Sensing.*

Kerle, N., Heul, S., Pfeifer, N. (2008). Real-time data collection and information generation usin airborne sensors. IN *Geospatial Information Technology for Emergency Response.* Zlatanova and Li (editors). (pp. 43 – 74). London, Taylor & Francis Group.

Laituri, M., & Kodrich, K. (2008). On line disaster response community: People as sensors of high magnitude disasters using internet GIS. *Sensors*, *8*(5), 3037–3055.

Li, J., Chapman, M. A.(2008). Terrestrial mobile mapping towards real-time geospatial data collection. IN *Geospatial Information Technology for Emergency Response*. Zlatanova and Li (editors). (pp. 103 – 122). London, Taylor & Francis Group.

Lukaszczyk, A. (2011). International experts blend space technologies and crowdsourcing to enhance disaster management tools. URL http://www.newswise.com/articles/international-experts-blend-space-technologies-and-crowdsourcing-to-enhance-disaster-management-tools

Meier, P. (2011). How to verify social media content: Some tips and tricks on information forensics. URL http://irevolution.net/2011/06/21/information-forensics/

Chen, K., Reed, C., editors (2012). Open GeoSMS Standard – Core. Open Geospatial Consortium.

URL http://www.opengeospatial.org/standards/opengeosms

Pautasso, C., Zimmermann, O., & Leymann, F. (2008). Restful web services vs. "big"' web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, (pp. 805–814). New York, NY, USA: ACM.

Pucher, A. (2009). Usability and implementation issues for cartographic information systems in interdisciplinary environments.

Schmitz, S., Neis, P., & Zipf, A. (2008). New applications based on collaborative geodata the case of routing.

Shirky, C. (2009). Here comes everybody : the power of organizing without organizations. Penguin.

Sutton, J. N. (2010). Twittering Tennessee: Distributed networks and collaboration following a technological disaster. In *Proceedings of the 7th International ISCRAM Conference*. ISCRAM.

Tornqvist, E., Sigholm, J., & Nadjm-Tehrani, S. (2009). Hastily formed networks for disaster response: Technical heterogeneity and virtual pockets of local order. In *Proceedings of the 6th International ISCRAM Conference*. ISCRAM.

Worboys, M.F. (1995). *GIS : A Computer Science Perspective*. London. Taylor and Francis.

Zhang, Y., Kerle, N. (2008). Satellite remote sensing for near-real time data collection. IN *Geospatial Information Technology for Emergency Response*. Zlatanova and Li (editors). (pp. 75 – 102). London, Taylor & Francis Group.

Appendix A: Saving geometry to PostGIS

The complete source code resides on Github: https://github.com/ndkv/gmer

Listing 1 shows a piece of the geometry saving JavaScript code. A HTTP GET request is send to the server through jQuery's $.get(server, data, callback function) function. The geometry resides in the geom variable (as Well-Known Text) which stores the result from the to_wkt() function shown in Listing 2.

```
function save_constraint(constraint) {
    var geom = to_wkt(constraint);

    $.get(server + "set_geometry/", {geometry: geom, user_comment:"empty.. "},
function(data) {

        var parsed = $.parseJSON(data);

        $("#status").html("Save: " + parsed.status);

        constraint.gmii_id = parsed.pk;
        constraint.gmii_version = 0
    });
}
```

Listing 1 Clients-side Javascript function that envokes an HTTP GET request containing the data to the server.

The to_wkt() function transforms the Google Maps Polygon object to Well-Known Text. The server side code is shown in Listing 3.

```
function to_wkt(constraint)
{
    var path = constraint.getPath();
    var path_length = path.getLength();
    var wkt = "POLYGON((";

    for (var i=0; i < path_length - 1; i++) {
        wkt += String(path.getAt(i).lng()) + " " + String(path.getAt(i).lat()) +
", ";
    }

    wkt += String(path.getAt(path_length - 1).lng()) + " " +
String(path.getAt(path_length - 1).lat()) + ", ";
    wkt += String(path.getAt(0).lng()) + " " + String(path.getAt(0).lat())+ "))";

    return wkt;
}
```

Listing 2 to_wkt function from Listing 1 that transforms the Google Maps polygon geometry into Well-Known Text.

The server runs on Python using the Django web framework. Django takes care of communicating with the database. To this end, desired data fields are defined as Python objects (see Listing 4) which Django automatically transforms into database relations and tables. Writing data to the database is now as easy as calling the .save() function of a Python object.

Once the geometry is received from the client as Well-known text it is unpacked and stored in the object's geom attribute (defined in Listing 4). In the current case, obstacles are versioned. In terms of database tables this results in a table that stores all obstacles and a different table that stores all versions. This is reflected in the code below by the obstacle = Obstacle() and version = Version() statements.

```
def set_geometry(request):
        geom = request.GET['geometry']
        creator_comment = request.GET['user_comment']

        obstacle = Obstacle()
        obstacle.creator_comment = creator_comment
        obstacle.save()

        version = Version()
        version.obstacle = obstacle
        version.version = 0
        version.date = datetime.datetime.now()
        version.geom = geom
        #obstacle.geom = 'POLYGON((0.0 0.0, 1.0 0.0, 1.0 1.0, 0.0 1.0, 0.0 0.0))'

        version.save()
        response = {"status":"Success!", "pk":obstacle.pk,
"version":version.version}

        #FUSION TABLES
        try:
            gft = FusionTables(version.geom.kml)
            gft.save()
        except AttributeError:
            print "GFT save failed";

        return HttpResponse(json.dumps(response))
```

Listing 3 Server-side Python code that saves the WKT to the database.

```
from django.contrib.gis.db import models

class Obstacle(models.Model):
    #obstacle_id = models.IntegerField()
    creator_comment = models.TextField(null=True)

class Version(models.Model):
    obstacle = models.ForeignKey('Obstacle')
    version = models.IntegerField()
    date = models.DateTimeField()

    geom = models.PolygonField()
    objects = models.GeoManager()
```

Listing 4 Definition of Obstacle and Version objects

Appendix B: Retrieving geometry from PostGIS

Retrieving geometries is based on the mechanisms explained in Appendix A: Saving geometry to PostGIS. Listing 5 shows the client-side JavaScript code that request a geometry. A GET request is sent to the REST endpoint at gmer.ndkv.nl/get_geometry. The data is sent as JSON. The obstacles are drawn by the draw_constraint() function as shown in Listing 6.

```
function get_constraints() {
    $.get(server + "get_geometry/", function(data) {
        $("#output").html(data);

        var parsed = $.parseJSON(data);
        var obstacles = parsed.objects;

        if (obstacles !== undefined) {
            $.each(obstacles, function(index, obstacle) {
                var new_constraint = draw_constraint(obstacle);
                new_constraint.gmii_id = obstacle.pk;
                new_constraint.gmii_version = obstacle.version;
                new_constraint.gmii_num_comments = obstacle.comments
            });

        } else {
            $("#status").html("The database is empty.");
        }
    }).error(function(request, error) {
            if (request.status === 0) { alert("Same origin policy?"); }
    });
}
```

Listing 5 jQuery AJAX GET-request that fetches data from the database.

```
function draw_constraint(obstacle) {
    var coordinates = obstacle.geometry.coordinates[0] // we do not expect holes
    var path = [];

    $.each(coordinates, function(index, coordinate_pair) {
       var lat = parseFloat(coordinate_pair[1]);
       var lng = parseFloat(coordinate_pair[0]);
       path.push(new google.maps.LatLng(lat,lng));
    });

    path.pop();
    return buildConstraint(path);
}
```

Listing 6 JavaScript code that drawes the retrieved geometry

Listing 7 shows the server-side code that retrieves the geometries. The geometry is retrieved in two steps: first the latest version of an obstacle is determined in line 3 after which a new query (line 6) is sent to fetches the latest geometries. The SQL equivalent of line 3 is

CREATE VIEW max_version AS (SELECT obstacle, MAX(version) as max FROM version GROUP BY obstacle);

The loop in which line 6 resides results in the following SQL query

SELECT * FROM version WHERE version.version IN (SELECT max FROM max_version WHERE obstacle = gmer_version.obstacle);

The result is then encoded in JSON and sent back to the client.

```
def get_geometry(request):
    geom = []
    max_versions = Version.objects.values('obstacle').annotate(Max('version'))

    for objects in max_versions:
        item = Version.objects.get(obstacle=objects['obstacle'], version =
objects['version__max'])
        comments = Comment.objects.filter(obstacle=item.obstacle.pk).count()

        geom.append({"version":item.version, "comments":comments, "pk":
item.obstacle.pk, "geometry":{"type": item.geom.geom_type, "coordinates":
item.geom.coords}})

    if len(geom) == 0:
        response = {"status":"Database is empty"}
    else:
        response = {"status":"Sucess!", "objects":geom}

    return HttpResponse(json.dumps(response))
```

Listing 7 Server-side Python code that retrieves the latest obstacle version from the database.

Reports published before in this series

1. GISt Report No. 1, Oosterom, P.J. van, Research issues in integrated querying of geometric and thematic cadastral information (1), Delft University of Technology, Rapport aan Concernstaf Kadaster, Delft 2000, 29 p.p.
2. GISt Report No. 2, Stoter, J.E., Considerations for a 3D Cadastre, Delft University of Technology, Rapport aan Concernstaf Kadaster, Delft 2000, 30.p.
3. GISt Report No. 3, Fendel, E.M. en A.B. Smits (eds.), Java GIS Seminar, Opening GDMC, Delft 15 November 2000, Delft University of Technology, GISt. No. 3, 25 p.p.
4. GISt Report No. 4, Oosterom, P.J.M. van, Research issues in integrated querying of geometric and thematic cadastral information (2), Delft University of Technology, Rapport aan Concernstaf Kadaster, Delft 2000, 29 p.p.
5. GISt Report No. 5, Oosterom, P.J.M. van, C.W. Quak, J.E. Stoter, T.P.M. Tijssen en M.E. de Vries, Objectgerichtheid TOP10vector: Achtergrond en commentaar op de gebruikersspecificaties en het conceptuele gegevensmodel, Rapport aan Topografische Dienst Nederland, E.M. Fendel (eds.), Delft University of Technology, Delft 2000, 18 p.p.
6. GISt Report No. 6, Quak, C.W., An implementation of a classification algorithm for houses, Rapport aan Concernstaf Kadaster, Delft 2001, 13.p.
7. GISt Report No. 7, Tijssen, T.P.M., C.W. Quak and P.J.M. van Oosterom, Spatial DBMS testing with data from the Cadastre and TNO NITG, Delft 2001, 119 p.
8. GISt Report No. 8, Vries, M.E. de en E. Verbree, Internet GIS met ArcIMS, Delft 2001, 38 p.
9. GISt Report No. 9, Vries, M.E. de, T.P.M. Tijssen, J.E. Stoter, C.W. Quak and P.J.M. van Oosterom, The GML prototype of the new TOP10vector object model, Report for the Topographic Service, Delft 2001, 132 p.
10. GISt Report No. 10, Stoter, J.E., Nauwkeurig bepalen van grondverzet op basis van CAD ontgravingsprofielen en GIS, een haalbaarheidsstudie, Rapport aan de Bouwdienst van Rijkswaterstaat, Delft 2001, 23 p.
11. GISt Report No. 11, Geo DBMS, De basis van GIS-toepassingen, KvAG/AGGN Themamiddag, 14 november 2001, J. Flim (eds.), Delft 2001, 37 p.
12. GISt Report No. 12, Vries, M.E. de, T.P.M. Tijssen, J.E. Stoter, C.W. Quak and P.J.M. van Oosterom, The second GML prototype of the new TOP10vector object model, Report for the Topographic Service, Delft 2002, Part 1, Main text, 63 p. and Part 2, Appendices B and C, 85 p.
13. GISt Report No. 13, Vries, M.E. de, T.P.M. Tijssen en P.J.M. van Oosterom, Comparing the storage of Shell data in Oracle spatial and in Oracle/ArcSDE compressed binary format, Delft 2002, .72 p. (Confidential)
14. GISt Report No. 14, Stoter, J.E., 3D Cadastre, Progress Report, Report to Concernstaf Kadaster, Delft 2002, 16 p.
15. GISt Report No. 15, Zlatanova, S., Research Project on the Usability of Oracle Spatial within the RWS Organisation, Detailed Project Plan (MD-NR. 3215), Report to Meetkundige Dienst – Rijkswaterstaat, Delft 2002, 13 p.
16. GISt Report No. 16, Verbree, E., Driedimensionale Topografische Terreinmodellering op basis van Tetraëder Netwerken: Top10-3D, Report aan Topografische Dienst Nederland, Delft 2002, 15 p.
17. GISt Report No. 17, Zlatanova, S. Augmented Reality Technology, Report to SURFnet bv, Delft 2002, 72 p.

18. GISt Report No. 18, Vries, M.E. de, Ontsluiting van Geo-informatie via netwerken, Plan van aanpak, Delft 2002, 17p.
19. GISt Report No. 19, Tijssen, T.P.M., Testing Informix DBMS with spatial data from the cadastre, Delft 2002, 62 p.
20. GISt Report No. 20, Oosterom, P.J.M. van, Vision for the next decade of GIS technology, A research agenda for the TU Delft the Netherlands, Delft 2003, 55 p.
21. GISt Report No. 21, Zlatanova, S., T.P.M. Tijssen, P.J.M. van Oosterom and C.W. Quak, Research on usability of Oracle Spatial within the RWS organisation, (AGI-GAG-2003-21), Report to Meetkundige Dienst – Rijkswaterstaat, Delft 2003, 74 p.
22. GISt Report No. 22, Verbree, E., Kartografische hoogtevoorstelling TOP10vector, Report aan Topografische Dienst Nederland, Delft 2003, 28 p.
23. GISt Report No. 23, Tijssen, T.P.M., M.E. de Vries and P.J.M. van Oosterom, Comparing the storage of Shell data in Oracle SDO_Geometry version 9i and version 10g Beta 2 (in the context of ArcGIS 8.3), Delft 2003, 20 p. (Confidential)
24. GISt Report No. 24, Stoter, J.E., 3D aspects of property transactions: Comparison of registration of 3D properties in the Netherlands and Denmark, Report on the short-term scientific mission in the CIST – G9 framework at the Department of Development and Planning, Center of 3D geo-information, Aalborg, Denmark, Delft 2003, 22 p.
25. GISt Report No. 25, Verbree, E., Comparison Gridding with ArcGIS 8.2 versus CPS/3, Report to Shell International Exploration and Production B.V., Delft 2004, 14 p. (confidential).
26. GISt Report No. 26, Penninga, F., Oracle 10g Topology, Testing Oracle 10g Topology with cadastral data, Delft 2004, 48 p.
27. GISt Report No. 27, Penninga, F., 3D Topography, Realization of a three dimensional topographic terrain representation in a feature-based integrated TIN/TEN model, Delft 2004, 27 p.
28. GISt Report No. 28, Penninga, F., Kartografische hoogtevoorstelling binnen TOP10NL, Inventarisatie mogelijkheden op basis van TOP10NL uitgebreid met een Digitaal Hoogtemodel, Delft 2004, 29 p.
29. GISt Report No. 29, Verbree, E. en S.Zlatanova, 3D-Modeling with respect to boundary representations within geo-DBMS, Delft 2004, 30 p.
30. GISt Report No. 30, Penninga, F., Introductie van de 3e dimensie in de TOP10NL; Voorstel voor een onderzoekstraject naar het stapsgewijs introduceren van 3D data in de TOP10NL, Delft 2005, 25 p.
31. GISt Report No. 31, P. van Asperen, M. Grothe, S. Zlatanova, M. de Vries, T. Tijssen, P. van Oosterom and A. Kabamba, Specificatie datamodel Beheerkaart Nat, RWS-AGI report/GIST Report, Delft, 2005, 130 p.
32. GISt Report No. 32, E.M. Fendel, Looking back at Gi4DM, Delft 2005, 22 p.
33. GISt Report No. 33, P. van Oosterom, T. Tijssen and F. Penninga, Topology Storage and the Use in the context of consistent data management, Delft 2005, 35 p.
34. GISt Report No. 34, E. Verbree en F. Penninga, RGI 3D Topo - DP 1-1, Inventarisatie huidige toegankelijkheid, gebruik en mogelijke toepassingen 3D topografische informatie en systemen, 3D Topo Report No. RGI-011-01/GISt Report No. 34, Delft 2005, 29 p.
35. GISt Report No. 35, E. Verbree, F. Penninga en S. Zlatanova, Datamodellering en datastructurering voor 3D topografie, 3D Topo Report No. RGI-011-02/GISt Report No. 35, Delft 2005, 44 p.

36. GISt Report No. 36, W. Looijen, M. Uitentuis en P. Bange, RGI-026: LBS-24-7, Tussenrapportage DP-1: Gebruikerswensen LBS onder redactie van E. Verbree en E. Fendel, RGI LBS-026-01/GISt Rapport No. 36, Delft 2005, 21 p.
37. GISt Report No. 37, C. van Strien, W. Looijen, P. Bange, A. Wilcsinszky, J. Steenbruggen en E. Verbree, RGI-026: LBS-24-7, Tussenrapportage DP-2: Inventarisatie geo-informatie en -services onder redactie van E. Verbree en E. Fendel, RGI LBS-026-02/GISt Rapport No. 37, Delft 2005, 21 p.
38. GISt Report No. 38, E. Verbree, S. Zlatanova en E. Wisse, RGI-026: LBS-24-7, Tussenrapportage DP-3: Specifieke wensen en eisen op het gebied van plaatsbepaling, privacy en beeldvorming, onder redactie van E. Verbree en E. Fendel, RGI LBS-026-03/GISt Rapport No. 38, Delft 2005, 15 p.
39. GISt Report No. 39, E. Verbree, E. Fendel, M. Uitentuis, P. Bange, W. Looijen, C. van Strien, E. Wisse en A. Wilcsinszky en E. Verbree, RGI-026: LBS-24-7, Eindrapportage DP-4: Workshop 28-07-2005 Geo-informatie voor politie, brandweer en hulpverlening ter plaatse, RGI LBS-026-04/GISt Rapport No. 39, Delft 2005, 18 p.
40. GISt Report No. 40, P.J.M. van Oosterom, F. Penninga and M.E. de Vries, Trendrapport GIS, GISt Report No. 40 / RWS Report AGI-2005-GAB-01, Delft, 2005, 48 p.
41. GISt Report No. 41, R. Thompson, Proof of Assertions in the Investigation of the Regular Polytope, GISt Report No. 41 / NRM-ISS090, Delft, 2005, 44 p.
42. GISt Report No. 42, F. Penninga and P. van Oosterom, Kabel- en leidingnetwerken in de kadastrale registratie (in Dutch) GISt Report No. 42, Delft, 2006, 38 p.
43. GISt Report No. 43, F. Penninga and P.J.M. van Oosterom, Editing Features in a TEN-based DBMS approach for 3D Topographic Data Modelling, Technical Report, Delft, 2006, 21 p.
44. GISt Report No. 44, M.E. de Vries, Open source clients voor UMN MapServer: PHP/Mapscript, JavaScript, Flash of Google (in Dutch), Delft, 2007, 13 p.
45. GISt Report No. 45, W. Tegtmeier, Harmonization of geo-information related to the lifecycle of civil engineering objects – with focus on uncertainty and quality of surveyed data and derived real world representations, Delft, 2007, 40 p.
46. GISt Report No. 46, W. Xu, Geo-information and formal semantics for disaster management, Delft, 2007, 31 p.
47. GISt Report No. 47, E. Verbree and E.M. Fendel, GIS technology – Trend Report, Delft, 2007, 30 p.
48. GISt Report No. 48, B.M. Meijers, Variable-Scale Geo-Information, Delft, 2008, 30 p.
49. GISt Report No. 48, Maja Bitenc, Kajsa Dahlberg, Fatih Doner, Bas van Goort, Kai Lin,Yi Yin, Xiaoyu Yuan and Sisi Zlatanova, Utilty Registration, Delft, 2008, 35 p.
50. GISt Report No 50, T.P.M. Tijssen en S. Zlatanova, Oracle Spatial 11g en ArcGIS 9.2 voor het beheer van puntenwolken (Confidential), Delft, 2008, 16 p.
51. GISt Report No. 51, S. Zlatanova, Geo-information for Crisis Management, Delft, 2008, 24 p.
52. GISt Report No. 52, P.J.M. van Oosterom, INSPIRE activiteiten in het jaar 2008 (partly in Dutch), Delft, 2009, 142 p.

53. GISt Report No. 53, P.J.M. van Oosterom with input of and feedback by Rod Thompson and Steve Huch (Department of Environment and Resource Management, Queensland Government), Delft, 2010, 60 p.
54. GISt Report No. 54, A. Dilo and S. Zlatanova, Data modeling for emergency response, Delft, 2010, 74 p.
55. GISt Report No. 55, Liu Liu, 3D indoor " door-to-door" navigation approach to support first responders in emergency response – PhD Research Proposal, Delft, 2011, 47 p.
56. GISt Report No. 56, Md. Nazmul Alam, Shadow effect on 3D City Modelling for Photovoltaic Cells – PhD Proposal, Delft, 2011, 39 p.
57. GIST Report No. 57, G.A.K. Arroyo Ohori, Realsing the Foundations of a Higher Dimensional GIS: A Study of Higher Dimensional Data Models, Data Structures and Operations – PhD Research Proposal, Delft, 2011, 68 p.
58. GISt Report No. 58, Zhiyong Wang, Integrating Spatio-Temporal Data into Agent-Based Simulation for Emergency Navigation Support – PhD Research Proposal, Delft, 2012, 49 p.
59. GISt Report No. 59, Theo Tijssen, Wilko Quak and Peter van Oosterom, Geo-DBMS als standard bouwsteen voor Rijkswaterstaat (in Dutch), Delft, 2012, 167 p.
60. GISt Report No. 60, Amin Mobasheri, Designing formal semantics of geo-information for disaster response – PhD Research Proposal, Delft, 2012, 61 p.


ZUIDPOLDER
VAN DELFGAUW
MOLENSLOOT
POLDER