UNIVERSITE DE GENEVE
Faculté des Sciences Economiques et Sociales
Département de Systèmes d'Information

GeoVTag: Trusting Virtual Tags

THESIS

Presented at Geneva University by:

Michel Deriaz

Thesis no 665
Geneva, 2008

Acknowledgements

I would like to thank the people that helped me, or played an important role, in the realization of this thesis.

The first person I'd like to thank is Professor Dimitri Konstantas, head of the ASG group of Geneva University. Since now many years Dimitri felt the importance and the potential of spatial messaging. He allowed me to explore in a deeper way his vision and now I can confirm that he was right; spatial messaging is a hot topic. It was really a pleasure to work with Dimitri: He can give clear and precise instructions and at the same time be sufficiently flexible to integrate any new idea. He trusted me for some projects even when he didn't really believed in them, and this trust proof resulted in some widely deployed applications as well as some new partnerships between the university and industrial actors.

The second person I'd like to thank is Dr Jean-Marc Seigneur, who works in similar subjects than me, but with much more experiences. Thanks to Jean-Marc I discovered the most reputable trust conferences, namely PST (Privacy, Security and Trust), iTrust and IFIPTM (International Federation for Information Processing Trust Management). We started by writing papers together and then little by little I specialized myself in my own domain. Finally I attended these conferences mainly in order to present my work to the best trust experts to be sure that my contribution is really a novelty that will improve the current state of the art.

The third person I'd like to thank is Dr Alexandre le Bouthillier, CIO of Organix IT, a company specialized in video surveillance, biometrics, and other security topics. His main contribution to this work actually only lasted a few seconds, but allowed me to save many months: He fixed me a deadline to finish my PhD. With the help of a motivating job offer, Alexandre encouraged me during the last few months.

The fourth person I'd like to thank is Mr. David Beni, CEO of arx iT, a company specialized in GIS (Geographical Information System). It is the industrial partner of a CTI project whose aim was to develop a tool to manage efficiently security boats during big events, like the internationally well known Bol d'Or competition in Lake Geneva. In short, David is the partner that allowed me to be financed during this work.

The fifth and last person I'd like to thank is my wife, Janique, who supported and encouraged me during the realization of this PhD. She took care about my well-being, spent some extra time with our children since I had to work some evenings and some week-ends, and gave me all the happiness that I needed to finish this work.

And finally I also want to acknowledge the financial support from the European Union project SECURE (IST-2001-32486) and the CTI project "GeoVTag: Trusted Geo-spatial Tags " (CTI P-No: 8643).

Contents

1 Introduction

Spatial messaging, also called digital graffiti, air graffiti, or splash messaging, allows a user to publish a geo-referenced note so that any other user reaching the same place gets the message. For example, let us consider the community of the Mt-Blanc mountain guides. The members would like to inform their colleagues about dangers in specific places. One guide publishes a geo-referenced message that informs about a high risk of avalanches, and any other guide that goes to the same place will get the warning, and comment it if necessary. Spatial messaging is a kind of blog in which editors and readers share the same physical place.

There are many reasons to believe that spatial messaging will become a wide spread concept in a nearby future. Today, people use the connection capabilities of their mobile phone mostly in one way, to download information. But in the same way that people passed from television to Internet, the next generation of users will probably become more "active" and create new content with their mobile phones. We already observe this tendency today for specific cases, like sharing pictures or videos recorded by mobile phones and published on some websites. If we remember how fast the computer power and the communication capabilities of these little devices increase, we can easily paint a glorious future for mobile technology.

We will see in the related work chapter that lots of non critical applications are already running on mobile technology. We insist here on the "non critical" aspect; it clearly implies that there is today no third party that proposes any serious application using virtual tags.

To our view the reason is simple: We cannot trust their tags. We do not talk about POI (Points Of Interest), data that is usually provided by a unique source and copied in the devices; this topic will be discussed more deeply later in this chapter. We talk here about virtual tags, pieces of information that can be posted by unknown users and modified by other unknown users. These virtual tags are posted in a collaborative way, like it is done in the Google Earth Community [1] where every user can post any geo-referenced information. But, we observe then that we cannot trust this information. Security tools are not sufficient; even if you can be sure about the identity of an author, it is useless if you do not know him and therefore cannot trust the content of his message.

So we arrive to the central point of this thesis. Since managing the trust information will probably be the most difficult part for many future applications using virtual tags, we designed a generic platform that provides a set of trust engines. The behavior of each engine can be personalized through rules and parameters in order to be adapted to a given application. We will see how and why our contribution improves the current state of the art, and we will also see that some applications, academic or commercial ones, are already using our work.

1.1 Main scenarios

This section describes three different hypothetical scenarios using spatial messaging. They help the reader to better understand the contribution of this concept to everyday life. Some of these scenarios will be cited as examples from other sections.

1.1.1 Signaling dangers in the mountains

Before they opted for a spatial messaging system, the members of the Mt-Blanc mountain guides' community faced some difficulties to inform their colleagues about dangers or other interesting events. For instance, how to inform them about a danger of avalanche at a specific place? Sending an SMS to all of them is difficult since the community is quite big and not very efficient since even the guides that are in a completely different place receive the messages.

Today, they use a spatial messaging system that allows them to post virtual tags where there is something to signal. Only the people that approach the tag area are now informed about the event. Let's now follow the trip of John, a very experienced mountain guide, expert in avalanches. John uses a mobile phone equipped with a GPS [64] in order to know his current position. Every ten seconds, John's application creates a virtual tag saying "John is here" and posts it at his current position. In case of an accident, the rescue team will be able to localize John easily.

During his trip, John finds a zone where he suspects a high risk of avalanches. After some measurements, he decides to create a virtual tag whose shape is a circle centered at his current position with a radius of 400 meters. Since 400 meters covers easily the dangerous zone, he defines that this tag is visible only while you are inside the area and that an alarm should be launched on your mobile device when you enter this area. He defines an expiration time so that the tag disappears automatically in 3 months.

A few moments later, John remarks that the shortcut to the village became slippery after the last snow fall. It remains safe during the day but becomes dangerous during nighttime since the path is quite narrow. He then shows a map on his mobile phone and creates a shape that more or less surrounds the path. This shape is the one of a new tag titled "Slipping path". He defines that this tag is visible from anywhere, but only between 18:00 and 6:00, and sets an expiration time in 3 days.

Jack, a friend of John, requests all the tags around his current position in a radius of 5 km and shows them on a map. He can therefore see that the shortcut has been tagged as sleepy, but does not see the danger of avalanches since this information is available only when you enter the visibility zone of the tag. He sees also some other tags posted by tourists. One of them says that there are wolfs in this area. Since Jack knows that wolfs always stay below 3000 meters of altitude, this tag posted at 3600 meters is probably spam. He therefore denies the tag, operation that decreases the trust value that Jack has in this tourist, and that marks on the tag that the user Jack doesn't agree with this tag. Later, when Jack enters the avalanche zone, he gets an alarm on his mobile device. Since he observes that the risk is real, he confirms John's tag. This increases the trust value he has in John.

The trust engine is then responsible to remove obsolete or false tags, for instance the ones that reached their expiration time or the ones that are denied by reliable people, to exclude malevolent users (people that try to remove true tags or that spam), and to return only "relevant" tags to the requesting users. For instance a user that confirmed some Jack's tags and therefore trusts him won't even get the tag mentioning wolfs, since Jack denied this information.

1.1.2 Tagging speed cameras

Many countries are currently installing lots of new speed cameras on the road. However their efficiency on security is far from convincing since there are more and more reports that show that drivers tend to brake suddenly when they see a speed camera (even if they are not speeding), which provokes traffic jams and accidents.

Michael, convinced that it is safer to observe the road instead of watching continuously the speedometer by fear of being flashed, opted for FoxyTag [26], a free and collaborative system to signal speed cameras on mobile phones. The idea consists in posting virtual tags close to speed cameras in order to warn the other drivers. These users will then get an alarm when they are closer than 15 seconds to a critical point, and a red point locating the speed camera appears on their screen. You can signal a fixed speed camera by pressing the key "1" of your mobile phone, a mobile one by pressing key "2" and that a camera disappeared (you get an alarm but you do not see any speed camera) by pressing "0". You are also invited to signal speed cameras that have already been tagged; by confirming their presence, you create trust links with other users and get more reliable information. The system excludes automatically users that do not vote "like the others". Roughly speaking, the more you participate, the more the information you get is reliable.

Since FoxyTag signals only speed cameras, the content of the tags is empty. When a user presses "1" on his mobile phone, the system takes the data provided by the GPS and creates a tag at the current position with no expiration time, and transmits also the current heading and speed of the user. The heading is used so that the tag is visible only for the users that are moving in the same direction and the speed helps the trust engine to compute with what precision this tag has been posted. Pressing "2" on the mobile phone simply adds a deadline of 6 hours to the current tag, so it is used to signal mobile speed cameras. And finally, pressing "0" indicates that we want to deny this tag. If other people deny this tag and nobody confirms it in a given delay, the tag is removed.

When he started using this application, Michael confirmed all the speed cameras he crossed. This allowed him to create trust relationships with the authors of the tags. So, when he requests the tags around him, the trust engine checks the trust values that Michael has in the people that confirmed and the ones that denied the requested tags. If some of them are unknown, the trust engine asks the friends (users trusted by Michael) and then the friends of the friends of Michael for advice. All the advices are then combined, and the trust engine decides whether this tag must be sent to Michael. If the trust engine makes the wrong decision, Michael will either deny or confirm the tag when he will cross the speed camera (if any), which will then update the trust values of the users in the tag's history. The more a user provides good contributions, the more he gets reliable information.

1.1.3 Reputation of restaurants

Richard likes good tables and goes regularly to new restaurants. He was used to check on the Internet if there are some comments on the targeted restaurant and to go only if the reputation was high, but he has been sometimes so disappointed that he asks himself if the comments are not simply posted by the owners of the restaurants. Another drawback of this way of doing was that it is not possible to know the reputation of the restaurants around him without knowing their name. For instance, when Richard visits a new place and decides to eat, he cannot easily see on a map all the reputable restaurants around him.

But now Richard uses a spatial messaging system and is much less often disappointed about a restaurant. When he finds himself in an unknown place, he uses his traditional navigation system to find all the restaurants around him. These are given as POIs (Points Of Interest) and are provided by the navigation system. They contain useful administrative information, like the opening hours, the phone number or the kind of food, but there is no information about the quality. That's why Richards always check if there are virtual tags posted at the same place. These tags are posted by previous clients that simply give their opinion about the restaurant.

When Richard started using this system he didn't know any of the other users and couldn't therefore trust the content of these tags. But, after each meal, he agreed or disagreed with the content of the available tags, which modified the trust values of the previous clients. And now, when Richard requests all the tags around him, the trust engine checks the opinion of the friends of Richard, of the friends of the friends, and so on until it is able to determine how reliable a given tag is in the eyes of Richard.

The trust engine that handles these tags is however quite different from the ones in the previous scenarios. In the previous scenarios users where tagging facts, for instance there is or there is not a speed camera. The job of the trust engine was therefore to exclude malevolent users that spam the system or that try to remove true tags. But in the current scenario the role of the trust engine is to create trust links between the users that share the same tastes. If Richard sees a tag where Horace mentioned that he didn't liked the food, it doesn't means that the food wasn't good. It only means that the advices of Horace mustn't count too much when computing the reputation of a restaurant for Richard.

A user can also give opinions for different things. Richard can for instance give a high recommendation for the food and the service, but a very bad note for the quality of the wines. A third user that does not drink alcohol will therefore probably like this restaurant.

Tag requests can also be personalized. For instance Richard can request only the tags that have been authored by Alice or the ones that have only positive ratings. More precisely, Richard can apply any filter he wants to his request. The trustworthiness of the tag is therefore only one parameter among others.

1.2 Additional scenarios

These additional scenarios won't be referred from other chapters and are only given here to help the reader to see all the opportunities where spatial messaging can improve every day's life. We therefore do not give technical details and do not explain how a trust engine can exclude malevolent users or create trust links between people sharing the same opinion.

1.2.1 Posting a picture

John is enjoying his vacations at Sunny Beach. He takes some pictures, adds comments about his tourist experience, then defines that all the people in his address book marked as "friends" have access to it, and finally post the whole at his current position. One year later a friend of John that chose the same place for his vacations finds the pictures and can read John's notes; he learns among other things that the expensive restaurant so recommended by the tourist guides actually does not worth it to go.

1.2.2 A virtual path for scuba-divers

"Do you already get lost during a diving tour?" Experienced divers like to say that either people answer yes, or they lie. Finding your way is very difficult in an environment with such a limited visibility, and a compass is only useful if you know where you are. We could of course paint big yellow arrows on the reef to define the diving tour, but Black Shark Diving Club found a better solution; they posted virtual messages along the path to follow. These messages can be empty, if their role is just to reorient the divers, or can also provide information about what has to be seen here. The diving computer of each diver indicates where the closest messages are, and allows also the diver to publish his own notes, intended for himself (for example to find his way back), or for his friends.

1.2.3 Helping visually impaired people

Blind and visually impaired people do not perceive their current environment like us. Sally, blind since her birth, knows very well the paths she follows every day to reach her school, an institute specifically build for blind and visually impaired people. Some new dangers, like road works, cannot be perceived in time by Sally. That's why her school decided to give the opportunity for each student to get and publish virtual messages (voice and/or vibrations), so that new dangers can be communicated in time to each other.

1.2.4 Road traffic management

Your life expectancy when you walk on the emergency lane of a highway is about 20 minutes, according to the French national police [2]. There are not such statistics for other kinds of roads, but we can easily believe that placing a warning triangle to signal a breakdown or an accident is a dangerous operation. The road traffic management system of Bob's car uses spatial messaging in different situations. In case of an accident, a virtual warning triangle is placed a few hundred meters behind the vehicle. If Bob gets stopped in a traffic jam on a highway, a message is placed before the last exit so that other drivers can reconsider their path. An unusual behavior that potentially represents a danger to others is also signalized by a spatial message. In all these cases, the spatial messages can either be confirmed by succeeding readers if they agree with the content, or repudiated if they think the message is misleading or outdated. The message is therefore automatically destroyed when too many users disown its content.

1.2.5 Follow me!

Samir, an Egyptian tourist guide, organizes every Wednesday a trip to visit the Bedouins. Since they are nomad, they post regularly a spatial message at their current position so that they can easily be found. During the trip in the desert, Samir always drives the first car and uses its strong driver experience to find a good path to the Bedouins, avoiding all the dangers like boulders or slopes. Every second a spatial message is automatically sent by his vehicle, thus drawing a virtual road. The tourist cars can then easily follows Samir's track, even if they have to leave a few hundred meters between each car to avoid the sand moved by the previous one. After the dinner at the Bedouin's, our procession can effortlessly find its way back thanks to the posted messages. And even Samir uses them, since driving at night in the desert is difficult and dangerous.

1.2.6 Virtual meeting points

It is not always easy to find friends in crowded or big zones, like an airport or a ski resort. To help people, we find more and more so called meeting points, meant to be easy to be found. But they are always limited in number (an airport has not hundreds of such points) or even absent (ski resorts). On ski holiday, Jack likes difficult runs when the rest of its family prefers easy ones. To satisfy every wish, they often split up and meet again a few hundred meters below. But sometimes they miss each other. In this case, Jack's wife creates a virtual meeting point at her current position. Jack can then easily rejoin his family.

1.2.7 Where is the nearest Italian restaurant?

The Boccalino, an Italian restaurant, posted a virtual message that covers a radius of 500 meters around the building. It is an advertisement for this restaurant which provides, among others, the price list and the number of available places. Mark, standing nearby and looking for an Italian restaurant, can now search via keywords all the advertisements about Italian restaurants in the neighborhood.

1.2.8 Safe toilets in Athens

Athens is known to be an interesting tourist site, but is also known for having dangerous toilets... It is indeed difficult to find public toilets that are safe. However, some tourist noticed that they can easily access to the toilets of some luxurious hotels, since the doorman would not dare to ask a possible customer of such an establishment to identify himself. A newcomer will therefore be happy to get all the neighboring tags signaling safe toilets as well as some instructions, like "The toilets are just on the left of the reception".

1.2.9 Distributing newspapers

Edipresse is responsible to distribute newspapers in a huge area. Drivers have to drop packets of newspapers at specific distribution points (kiosks, halls...). Even if the itineraries and the affected driver change dynamically according to external parameters (weather, traffic jam, newspaper bigger than usual...), it remains still easy to find the distribution points thanks to Edipresse's system. This system uses a conventional navigation system (like TomTom [63]) to bring the driver close to the distribution point, and then it uses virtual tags to describe the exact position. Some tags are arrows used to direct the user, and some are pictures which show the exact position where to drop off the packet of newspapers.

1.3 Technically it's easy

Posting and getting virtual tags is technically very easy. A mobile phone containing a positioning system, or coupled to it, is sufficient. Posting a tag consists roughly in sending a message and attaching the current position to it. To get the virtual tags around him, the user simply sends its current position to the server, which answers with all the messages that have been posted in a given radius around the given position.

There are many ways to obtain one's current position. The most well-know is the GPS (Global Positioning System) [64], an American system made of 24 satellites equipped with a very precise clock and that send the current time to the Earth. A GPS receptor compares the different signals it catches and determines its current position. This system works everywhere on Earth with a precision between 5 and 50 meters, but degrades in difficult situations (for example a place surrounded by high buildings), and becomes useless in many indoors places. The GPS system will in a few years be completed by Galileo [65], a European system using 30 satellites and working in a similar way.

But there are also local solutions, like wireless triangulation. It consists in analyzing the different signals strengths, and according to the positions of the antennas, determining the position of the device. But this method is less precise (in the best conditions we estimate to have about 50 meters, but much more in worse conditions) than GPS or Galileo.

We observe that "positioning" is an active research topic, and that mobile phone operators stay in the run by preparing (and already providing for some of them) the aGPS technique. Assisted GPS (aGPS) is a signal sent from the antennas of the mobile phone operator that helps to localize the satellites. The user gets then a position much faster.

However, the technical solutions used to obtain the current position are outside the scope of this thesis. We just assume to have a technical manner to get a position, expressed in latitude and longitude, and that the precision as well as the availability of this information can vary according to time and position. We also assume that we know with what precision a positioning device deliver its information. We can therefore completely abstract from the techniques underneath in this work.

1.4 Virtual tags are not POIs

Different places exist where you can share your POIs (Points Of Interest) with other people, like POIPlace [11] or the Google Earth Community [12], but you can easily convince yourself that this information cannot be trusted. When you search a specific feature, like the "Jet d'eau" of Geneva, you will find it located in several different places. POIs suffer from the following drawbacks:

- They can be trusted only if they come from a known and reliable source. Everybody can, intentionally or not, publish wrong or misleading information. To deal with that, we need to know and trust the provider. There are people paid by companies (like Tele-Atlas [66]) just to drive and record points of interests. This information is usually sold and not always up-to-date.
- No creation time: The time component is not part of a POI. A reader has therefore no idea if the POI is still current or if it describes a feature that perhaps disappeared a long time ago.
- No expiration time. It is not possible to mention a temporary event with a POI.
- No information about the precision of the position. A position is usually expressed in latitude and longitude, typically with 6 digits after the decimal point. This guaranties a precision of less the 0.12 meters worldwide. However, the devices that are used to record these positions are probably less precise. Typically, a GPS device has a precision between 5 and 50 meters. And this difference can be critical in some situations.
- No possibility to comment them. A POI is static information and there is no way to comment one, or to ask to remove it because it is out of date.
- Search possibilities are limited to position and category. Since the author is unknown, we cannot search tags posted by a particular person or request only the ones that are posted by friends.

In this thesis we define virtual tags and go a step further. A virtual tag can contain any type of information (sounds, video...), can be reviewed and commented, is aware of the time component, and maintains trust information. Since everybody can publish and make reviews, it favors lots and up-to-date tags. The trust engine is then responsible to exclude malevolent users, but also to create trust links between the people sharing the same opinion. We will define later in this document more precisely what a virtual tag is, and what fields we can expect to find in all of them. But we will also see that an application developer can propose some extensions in order to adapt them to a specific case.

2 Related work

At least to our knowledge, we are the first to study the trust aspects in spatial messaging. Actually, even if we type only "spatial messaging" in Google [67], the first results point directly to our former papers. We find also some people that use our definition to describe it, like for instance in the alvafilm website [68]. So if we add the trust component to spatial messaging, we reduce even more the chances of finding some parallel work.

Since we couldn't find any similar work, we divided this chapter in three parts. The first part describes other work done for spatial messaging. The second part gives a state of the art in the trust domain. And finally, since we used a speed cameras warning system in order to test our models, the third part gives a list of other warning systems.

2.1 Spatial messaging

Before starting this section, we would like to precise the difference between spatial messaging and LBS (Location Based Services). In short, LBS is a kind of spatial messaging in which the user can only **get** data, and **not post** it. Lots of LBS applications for augmented cities (tourists get information, in their mother tongue, about their current place) or augmented museums (visitors get information about what they are looking) have already been implemented. We are clearly interested in spatial messaging in general, where users also post information.

2.1.1 E-Graffiti

E-Graffiti [3] is a spatial messaging application that allows a user to read and post geo-localized notes. These notes can be either public or private, meaning that only the set of people defined by the author are able to read the note.

E-Graffiti has been designed to study the social impacts on spatial messaging. 57 undergraduate students were given a laptop with E-Graffiti for a semester. All their activity has been logged and studied. And the results are far from encouraging. At the end of the semester, it came out that a user logged into the system only 7.6 times in average (std dev: 12.6), and that actually most of the users stuck to initial test messages. Another disappointment was that most of the posted notes were not related to their position. For example, a number of people posted notes to advertise a website. The system was designed so that the user could only get messages available at his current position, but it was possible to post a new message at any place from anywhere.

Technically, the position of the user is determined by the wireless access point to which the device is connected. The precision is therefore limited to the building in which the user is.

2.1.2 GeoNotes

GeoNotes [4] has more functionalities than E-Graffiti. While posting a note, the user can choose how he is going to sign it (for privacy reason the user can write any text he wants as a signature), decide whether people are allowed to comment it, and decide whether anyone can remove this message. For the readers, the graphical interface of the application provides some interesting functionalities like showing all the neighboring messages or sort them according to different criteria. Inspired by the E-Graffiti evaluation, GeoNotes discarded the remote authoring of tags as well as the possibility to "direct" notes to certain users.

The main interest of the GeoNotes authors seems to be the navigation problems in the virtual messages space. How to find a specific note? How to select only relevant messages? One answer of these questions consists in giving to the readers the possibility of ranking the notes. Each user maintains also a friends list, which can be used as a filter. But the trust and security aspects have not been taken into account. It is easy to usurp someone's identity and post false notes. An analysis of a GeoNotes log made during a real-use study showed that 6% of the messages have been signed using someone else's identity.

2.1.3 ActiveCampus Explorer

ActiveCampus Explorer [5] goes a step further by displaying also where other users are. Every user holds a PDA and its location is determined by comparing the signal strength of different wireless access points. Thus, the system knows the position of all its users, and communicates this information to the all of them that are close together. Like E-Graffiti and GeoNotes, it is also possible to tag objects.

2.1.4 Socialight

Socialight [17] allows a user to post some data to a specific place, intended for himself, for his friends, or for everybody. Meta-data containing keywords and geographical coordinates are attached to the posted data, in order to facilitate searches. Tags are called "Stickyshadows" and can be viewed with some specific mobiles phones (and equipped with a positioning system) via the Socialight Mobile application, or by browsing the Socialight website. A nice feature they provide consists in showing Stickyshadows on maps.

2.1.5 Context Watcher

Context Watcher [16] is a mobile phone application written in Python for Nokia Series 60 based on the MobiLife framework [19]. The first version of this application already uses the notion of confirmed buddy for security and trust purposes. They have a part that they called trust engine in their architecture but a closer look to it shows that it is actually only an access control system. Policies and profiles are use to decide who can access what data and under what condition, but there is no trust mechanism that informs how reliable a requested information is.

2.1.6 Summary

These projects don't seem to be successful. E-Graffiti and GeoNotes have been abandoned a few times after their launch. Socialight is still active, but there are seldom new posts. We believe that the lack of success is related to the lack of interest... in publishing notes just for publishing notes! Spatial messaging would probably have more chance to emerge if we focus on specific communities, with real problems that could be solved by this concept, rather than imposing the system to students without giving them any good reason to use it. But then we need a trust mechanism to exclude malevolent users. In GeoNotes people may stay anonymous, but we saw that user then usurped others' identities; it is therefore not possible to trust a message. In E-Graffiti users reveal their real identity, but it useless to know that a message has been posted by a certain "John" if you do not know John.

Commercial systems usually implement all the conventional security tools (username, password), but there is no trust engine that informs about the reliability of a given message. It means that it is always a human that plays the role of the trust engine and that excludes what he things are malevolent users.

However, in widely deployed systems (like for instance our FoxyTag [26] application that informs about speed cameras in all Europe) where there is only very little human interaction, only a trust engine can ensure a high quality of the data.

2.2 Trust

Trust is a very active research domain. It started by providing solutions for centralized systems (for instance the reputation system in eBay where seller and buyer can rate one another after a transaction), and then quickly switched to peer-to-peer systems. Peers rate each other and the combination of all the values informs about the reputation of the peer. The challenge here is where to store trust values, as there is no central server. Among the proposed solutions, we mention here a few of them:

- In EigenTrust [55] each peer has a set of mother peers responsible for storing its trust value, and therefore each peer acts also as a mother peer for others. It resists to an attack even when up to 70% of the peers are colluding in order to subvert the system. Peers are anonymous.
- An interesting system that is similar to EigenTrust, but in which peers store their own trust value locally, is called Elicitation-Storage [56]. The Elicitation-Storage protocol is used to protect cryptographically the trust value. The requester gets the IP address of the former requesters and checks with them the authenticity of their vote.

- The Secure project [57] aimed to describe in a formal way what trust is, staying as close as possible to the human notion of trust. The motivation for the project was that the number of entities in Internet systems is becoming very large. Consequently, it was important to develop security models that allow nodes to measure the risk involved in interacting with other nodes that they have not met before. The secure project implementation has been tested with a mail application: A proxy between the peer and his mailbox was analyzing the behavior of the user (for instance if he moved a message in his spam folder) and updated the trust values according to it. The reputation system allowed the different peers to share their information in order to exclude faster the spammers.
- Kinateder and Rothermel [58] present a peer-to-peer system that provides trust and recommendations about different categories of topics. Similar to sites like Epinion.com [53] or the rating system that we find in eBay [54], but peer-to-peer.
- The TrustMe protocol [59] builds trust in peer-to-peer networks. The trust value of a specific peer is anonymously stored on another peer. Communications are encrypted using sets of private/public keys. The drawback is that all peers have to connect to a bootstrap server when they join and when they leave the network (in order to transmit the hosted trusted values to another peer).
- Anwitaman Datta, Manfred Hauswirth and Karl Aberer present in [60] how P-Grid can be used to implement a distributed PKI (Public Key Infrastructure), enabling c2c (customer to customer) services like eBay but without any centralized system. Unlike PGP that uses the web of trust approach to access a particular public key, this system uses a statistical method; many peers are queried, and the information is rejected if a quorum a peers cannot be obtained.

Very interesting and promising decentralized solutions like the EigenTrust algorithm made the community to forget one aspect that is only seldom taken into account: time. In practice time is important. Someone you trusted a long time ago is perhaps not trusty anymore. Even people with a very high reputation can become malevolent afterwards. Since in human communities the trust is very time dependent, we believe that this component should also be included in trust engines and particularly in the spatial messaging context where posted information can simply become obsolete after a while.

Guha [16] built a generic trust engine allowing people to rate the content and the former ratings. He recognized however that in case of highly dynamic systems (like in spatial messaging where tags can appear and disappear very quickly), "Understanding the time-dependent properties of such systems and exploiting these properties is another potentially useful line of inquiry." Most existing trust metrics update their trust values only after a specific action, like a direct interaction or the reception of a recommendation. The few trust engines that take the time component into consideration simply suggest that the trust value decreases with the time. Mezzetti's trust metric [17] consists in multiplying the trust value at time *t* by a constant between 0 and 1. In Bayesian-based trust metrics [19, 20], the trust value converges to its initial value over time. All these models work in situations where the changes occur slowly, but are challenged in short-lived cases.

Unlike the spatial messaging community that seems to be less and less active, the trust community seems to grow and commercial applications are more and more interested in their work. We find for instance some attempts to add trust in Wikipedia [40] articles, like it is presented in a paper from Pierpaolo Dondio, Stephen Barrett, Stefan Weber and Jean-Marc Seigneur [69]. However, we haven't found yet any work on trust in the spatial messaging domain.

2.3 Speed cameras warning systems

As the number of speed cameras increases on European roads, we find more and more services that help the drivers avoiding expensive pictures. We will talk neither about illegal means (for the majority of European countries), like the radar detectors provided by RadarBusters [42], nor about non-technical means like phone centrals providing vocal information. We will concentrate here only on information systems that inform drivers about speed camera positions, which is completely legal according to the law of most European countries.

2.3.1 Mogoroad

Mogoroad [62] is a well-known system in Switzerland to announce traffic perturbations, police controls, and of course fixed and mobile speed cameras. It works on most mobile phones. They collect their information from different partners, like radio stations and newspapers, as well as from their own users that can either signal an event by phone or through an application running on mobile phones. There is no trust engine to validate the quality of the data. According to their CEO, Roberto Marra, it is the experience of the employees that collect the data that is used to differentiate useful and correct information from spam. In practice this works quite well since the covered area is small. However, such a system could not easily be extended to work worldwide while providing the same quality of information. The cost of this service is (in 2008) about 110 € per year.

2.3.2 SmartSpeed

SmartSpeed [43] is an application running on Windows Mobile that informs the driver about dangerous zones, traffic jams, and speed cameras. Working with all NMEA [41] compatible Bluetooth GPS, the program compares the current position with the "events" to come and informs the user through a voice synthesizer. Maps and "events" files can be downloaded in advance, and a GPRS [73] connection allows the user to get recent information. An interesting functionality allows any user to send a new event to the server, which will in turn inform all the users. A typical use consists in signaling mobile speed cameras to other drivers. Even if presented differently, it is clearly a way of doing spatial messaging.

The light version a SmartSpeed is relatively cheap (30 € including free updates for one year) if you possess already a smartphone and a Bluetooth GPS. However, messages sent by other users to signal mobile speed cameras are not verified and are available only for one hour. And users are not really motivated to post such messages since they have nothing to gain in signaling a new "event". SmartSpeed seems more adapted to signal fixed speed cameras than mobile ones.

2.3.3 Coyote

Coyote [44] is an independent system sold as a little box containing a GPS. When the driver approaches a speed camera, Coyote informs him orally about the remaining distance to this camera. To signal a new speed camera (or a new position for a mobile one), the user can simply press once the button on the top of the box. To signal a speed camera on the opposite direction, the user presses twice the button. This information is then sent to the server thanks to an included GPRS card, where a human operator verifies (previous messages of that user, comparison with other users, using another speed camera information service...) the plausibility of the information before broadcasting it to all users.

Despite it is very simple to use, Coyote remains an expensive system (699 € for 2 years with unlimited use and including communication fees) that not everybody can afford. And if there are too few users, then the chance that **you** are the first that discover a speed camera (by being flashed!) is high...

2.3.4 InfoRad

Autonomous and easy to use, InfoRad [45] beeps when the driver enters a "risky area". All the risky areas, materialized with a speed camera, are stored in the on-board database. It works thus only with fixed speed cameras and it is not possible to signal a new one to other drivers. It allows however a user to add its own risky areas for personal use. Their website provides time to time updates of risky areas. The device with an unlimited access to their database costs about 200 €.

2.3.5 POIplaces

A POI (Point Of Interest) is a geo-referenced item that presents a particular interest, like a restaurant, a fuel stations, or a car park. Written in standard formats, POI lists can be used by most navigation systems. POIplaces [11] is a website where people can share their own POIs. One successful topic is speed cameras. In the same way than for restaurants or fuel stations, users can download for free the list of all speed cameras. Their navigation system can then be configured to emit a sound when they approach a POI.

Free for everyone that owns already a GPS and a navigation device, this solution is however far from perfect. Since everybody can publish his POIs without any control, the speed cameras database is incomplete (lots of speed cameras are missing), redundant (several POIs for the same speed camera), incoherent (speed cameras have been found in a forest...), and mobile speed cameras are not taken into consideration.

2.3.6 GpsPasSion

GpsPasSion [46] provides active forums about the different topics of the GPS world. Some of them are specialized in the speed cameras domain and aim to collect information about their positions. They provide time to time an update of their POI file that can be freely downloaded. Compared to POIplaces, the list is smaller (lots of speed cameras are missing), but is more consistent since they check the information before updating their list. Members that submit new positions have also access to a list containing the preferred places for mobile speed cameras.

3 What is trust?

The Merriam-Webster English dictionary defines the trust as "assured reliance on the character, ability, strength, or truth of someone or something". But everyone has its own definition of the word "trust". While in general trust refers to an aspect of the relationship between individuals, the term has a completely different meaning depending on the domain it is used. In sociology, for example, we say that a person trusts someone else if he accepts to rely on him. In law, trust is a legal arrangement in which one person manages the property on behalf of another. A trust company is a financial institute, most often a kind of bank. In computer hardware security, trusted computing refers to chips or devices which the user is forced to trust; vulnerabilities in such a component can compromise the overall security of the computer. In a higher level of computer security, trust is a mechanism which aims to connect together entities that behave as expected. In cryptography, trust is the level of certainty that a public key belongs to the correct owner.

All the above definitions are in no way "official" ones. Even if we reduce our scope to computer security, we notice that different authors have also different definitions of what trust is.

This chapter presents our own point of view. We start by comparing how humans and machines are handling trust today. We then propose a little scenario supposed to take place in a nearby future in order to show how we think trust applications will evolve the next years. After that we present what a trust model is in order to help the reader to imagine how this information is stored and managed by a computer. In section 3.4 - "Trusting spatial messaging" we see why traditional trust models cannot simply be applied to spatial messaging and give the main points we will have to take into consideration. We try to stay as close as possible to what most people would intuitively describe as being trust, and therefore we inspire ourselves on how humans create trust relationships between them. Finally, we conclude this chapter by studdy the relation between trust and security.

3.1 Differences between computers and humans

One trusts someone or something if he or it behaves as expected in a given context for a given situation. Computers often use a very simple trust policy. Anyone that is able to give a username and the corresponding password is considered as trusty. A swindler that gets your password can enjoy freely of all your privileges, as long as he wants. A strange or unusual behavior, like installing key loggers or deleting system files (instead of typically checking for email as soon as the computer starts), won’t alarm the system. Once you are considered as trusty (correct password), you remain trusted until you log out.

However, humans use a more sophisticated trust policy when they interact with each others. When you go to a new doctor, you probably trust his competences just because you see the letters “Dr” on his badge. But if your health deteriorates after some visits, you become suspicious and you want to revaluate the trust value you previously accorded to this doctor. You need a stronger proof of his competences. Perhaps you will search information about his global reputation, by asking a doctors association (do they know him?), or by checking if his name appears somewhere in the yellow pages. Or you will ask your friends if they know about him. We see clearly that the intuitive human trust policy is much more complex than the static and one-time-check policy typically used by computers. Humans use dynamic trust values, computed according to their own observations, past experiences, global reputation, and recommendations given by trusted parties, like friends.

3.2 Trust policies in the future

People use more and more electronic devices to assist them in their tasks or for entertainment, but no network is built between devices without direct or indirect human interaction. A transfer of a business card from one PDA (Personal Digital Assistant) to another necessarily requires a manipulation from both end-users, even if the operation is actually very simple. The emergence of mobile computing will probably bring some big changes in the way that computer networks are built and evolve with the time. Trust policies will be affected by these changes. The following hypothetical scenario, taking place in a nearby future, gives a first approach of how we imagine day-to-day live modified by these technological advances.

3.2.1 Scenario

Like a majority of his friends, Martin is wearing at his wrist a small electronic device, called by use an IPDA (Intelligent Personal Digital Assistant). Not bigger than a conventional watch, it is the kind of device that people are wearing all the time. It records as many information as possible about its owner, like preferences and behaviors adopted in the different experiences of his life. After a while, the IPDA knows its owner very much, and is therefore able to autonomously take decisions.

Martin is working as a security consultant and has to travel a lot in order to make conferences and organize classes. His diary has to be very flexible, because some emergencies (like a successful hacker's attack in a company) can loom up suddenly and oblige Martin to reorganize his planning. It is a time consuming task that would borrow Martin's mind (for example not to forget to rent a car, make sure that the conference room will be fitted out with all the needed devices, ...) if he could not just rely on his IPDA to do this job according to his requirements and preferences.

After a phone call, Martin asks verbally his IPDA to organize him a conference in Berlin. He indicates also that it is a high priority request (80/100) and that he feels quite tired. The IPDA re-organizes the planning, cancels the low priority tasks, and because Martin is tired, will book a flight not too early in the morning even if the price is a little bit higher. The IPDA organizes the whole trip (travel, hotel, conference room, cars, restaurants, information mail for his colleagues...) and asks Martin's attention only for crucial decisions that cannot be computed without a reasonable chance of choosing the best solution.

The main idea is that what we call Ambient Intelligence Devices (AID), like IPDA, flight booking systems, or hotel reservations, are able to communicate with each other. For instance, a hotel will not only provide a web site for online booking, but also an AID interface allowing others AID to communicate with it. Every object that is able to communicate in an Ambient Intelligence System is called an AID. We can also imagine that a radiator asks about the preferred temperature of a client directly to his IPDA. The main difference with traditional web services is that an AID understands semantics. It is therefore possible for an AID to communicate with another AID even if they have never met before, and if they do not know exactly what functionality they offer to each other. In traditional web services, a human interaction is necessary to bind two or more of them together according to their API; AIDs devices are able to do it alone.

After landing in Berlin, Martin’s IPDA guides him to his car, opens it (the AID of the car recognizes the digital signature of the client's IPDA), and computes the best itinerary to the hotel. Once arrived, Martin decides he wants to eat Chinese tonight. His IPDA finds in its database that a good friend of Martin, Jordan, who loves Asian food, often travels in Berlin. It asks this friend's IPDA for advice. The "Peking" seems to be one of the bests. Martin's IPDA then connects to the restaurant in order to get the opening hours and a price list. It trusts more Martin's friend about the quality of the food, but trusts more the restaurant about administrative information that can change over the time. The last pieces of information are gathered from a web site that registers traveler comments about good and bad experiences they lived during their journey. After computing a different trust value for each source of information, a final calculation confirms that the "Peking" will probably be the best restaurant for Martin at this time.

Traveling seems to make Martin very hungry. He ordered the “Peking menu” made of two starters, one main course and a dessert. Once full, he tells his IPDA that he is really happy with this restaurant. This information is recorded as a pleasant experience and a good tip is computed. Every IPDA runs an e-purse software, and payments are made with e-cash.

The next morning, Martin gets up and, after breakfast, turns on the terminal that is in his room. For security reasons (Trojan horses), there are no hard-disks on such devices, clients use their own one which is more and more often their IPDA. Martin checks his e-mails, the latest news and then makes some final corrections to the slides he wants to present during his conference which begins in two hours. To avoid stress, Martin's best remedy is shopping. He lets his IPDA to direct him towards the nearest shopping centre, and then heads for the music department. After a while, Martin is reminded (from his IPDA) that time is running out. He heads for the exit, catches some chocolate on his way, and walks through the payment gate. Martin does not need to queue up at the traditional till. Every item is labeled with a RFID (Radio Frequency IDentification [74]) tag and payment is made automatically by the e-purse when its owner walks through the payment gate.

At the bus stop, Martin gets tempted by having a coffee. His IPDA consults the virtual notice board attached to the coffee machine. It is very common for such kind of machines to have a virtual notice board, on which users' IPDA write their experience. In our case, we observe that the machine is working on average only 50% of the time. But we see also that the last four clients enjoyed a positive outcome. According to this and to the limited amount of risk involved, Martin's IPDA accepts the financial transaction without asking its owner's authorization.

Relaxed, Martin joins the conference room, launches the welcome slide of his presentation using the speaker's terminal, and goes to the main door in order to personally greet every participant.

3.2.2 Scenario discussion

Let us consider the following points (hypothetical):

- There are different trust values for the same person. Martin accords different trust values for his friend Jordan. He gives 80% when they are talking about Chinese food, but only 30% when the subject is politics.
- Trust is transitive. Martin gives 70% of trust (about Chinese food) to the former girlfriend of Jordan, despite the fact he never saw her. He just knows that Jordan accorded 90% of trust to her for that topic, and therefore he computed his own value of trust.
- Trust evolves with time. If Martin is happy about a restaurant, he will increase the trust to the devices that recommended this place, or decrease it if he comes out disappointed.
- Decisions are not only taken regarding trust. The risk of every action is computed and combined to the trust value. The risk represents actually the maximal cost of an operation that fails. The context is very important in the evaluation of these values. For example Martin accepts to go in a restaurant if the chance of being happy is 75%, but he refuses to make a financial transaction if the chance of not being hacked is also 75%.
- More importance is given to recent events. In the example of the coffee machine, Martin's IPDA accepted the transaction despite the fact that the machine works on average only 50% of the time. Intuitively, if it worked well a few minutes ago, then the probability that it will work well once more is high.

As machines imitate human social behaviors, it seems clear that trust policies will also have to follow this tendency. We need policies that can be used between machines themselves as well as between humans and machines. Like human to human interactions, our devices are meant to meet unknown ones. Trust will be built and constantly updated according to the outcomes of these interactions

3.3 Trust models

Trust is well understood by humans, but seems to be very difficult to model. We can of course take a very simple model in which we compute the trust value *T* like:

$$T = \frac{m}{m+n}$$

Where we compute *m* and *n* as follows:

$$m = |PO|$$
$$n = |NO|$$

$|PO|$ is the number of positive outcomes and $|NO|$ the number of negative outcomes. But this first model does not take into account the time. We will try to modify our simple model in order to give more importance to recent events (like humans would do) and to give more importance to dispersed events (a regrouping of negative outcomes could be a temporally breakdown). We propose to compute *m* and *n* as follows:

$$m = a \cdot |PO| + \frac{b}{T} \sum_{i=0}^{i=|PO|} t(po_i) + c \cdot |PO| \cdot \delta_p$$
$$n = a \cdot |NO| + \frac{b}{T} \sum_{i=0}^{i=|NO|} t(no_i) + c \cdot |NO| \cdot \delta_n$$

Where:

- *a, b* and *c* are parameters in [0..1], with a default value of 1. The value *a* defines how important the outcomes are regardless of the time, the value *b* defines the importance to give to recent events compared to old ones, and the value *c* defines how important the dispersion of the outcomes is.
- *T* is the current time, expressed in the number of seconds elapsed since a reference time.
- $t(po_i)$ is the time when positive outcome *i* occurred.
- $t(no_i)$ is the time when negative outcome *i* occurred.
- δ_p is the dispersion of the positive outcomes.
- δ_n is the dispersion of the negative outcomes.

The dispersion can be computed by different ways. Our solution consists in computing, for each point, the distance to the average position of the points that are on the left (points lower than average) and the distance to the average position of the points that are on the right (points above average). Dividing the sum of these two distances by the distance that separates the two extreme points gives the dispersion.

Figure 1 shows an example of positive outcomes (PO) and negative ones (NO). It represents the virtual notice board of the coffee machine (see the former scenario).

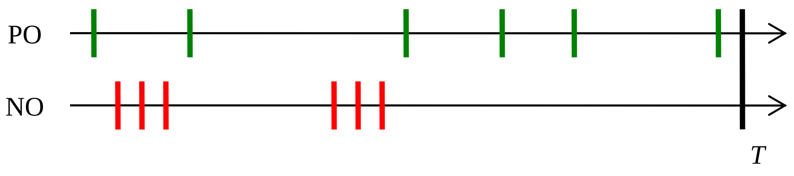


Figure 1. Timeline with positive and negative outcomes

Since there is the same number of positive outcomes than negative ones, our first basic model gives a trust value of 50% (figure 1). But we would probably intuitively give a higher trust value. Firstly, because the last outcomes are positive. Secondly because the dispersion of negative outcomes is much smaller; it means that the machine sometimes falls down, but that it works quite well otherwise.

Our new model takes care of these observations. In the computation of *m* and *n*, the second term gives more importance to events that occurred near *T*, and the third one gives more importance to event that are scattered.

If Martin's IPDA uses this trust model, it will accept the transaction because the last outcomes where positive and scattered. If the machine was correctly working for the last few minutes it should probably also work now (at time *T*).

It is clear that this model is only a simple example. As we wrote it some sections above, the trust policy used by humans is very context dependent. We notice also that this model does not take into account the risk that is involved. However, by choosing correctly the different parameters *a*, *b* and *c*, we can still fit to some every day life situations.

3.4 Trusting spatial messaging

Lots of work has already been done in the trust context (see 2 - "Related work"), and the question that arises is why not just using well-known trust models and apply them to virtual tags? The answer is simply that it will not work. Indeed, traditional trust models are mainly designed with file sharing or auction applications in mind. In this case, people are rating each other and when user *A* wants to download a file (or buy an item) from user *B*, he questions the system in order to determine how trustworthy user *B* is. Currently, commercial systems (like eBay) are using very basic centralized systems, and the academics are suggesting solutions to transform such systems into peer-to-peer architectures.

But spatial messaging is noticeably different from file sharing or auctioning and needs therefore a new trust model. The key difference is that in spatial messaging it is difficult to increase its own trust without making a significant contribution. For instance, to post a new tag that will be confirmed by others (in order to create a trust link), a user will have to be physically there (to make the observation that deserves a tag). In a similar way, the user that deletes an outdated tag makes also a significant contribution. So, even if a user wants to increase his trust value in order to harm the system later, his former contribution will compensate his future bad behavior. And this is an interesting difference that will be used in this work in order to construct trust engines. It is, at least to our knowledge, a novelty in the trust domain and can be considered as the key point of this work. In "traditional" trust systems, it is always possible to easily increase one's own trust value in order to subvert the system later. For instance, it is easy to sell honestly a few goods in eBay in order to increase ones trust value. It is also easy to provide a few good files in a file-sharing system and then use the resulting good reputation to send Trojan horses. But in spatial messaging, a user can increase his trust value only in return of a significant contribution. We will also see in 3.4.3 - "Updating trust values" how we can make it impossible for a user to switch regularly between good and bad behavior in order to keep a minimum trust value, and how to avoid that a user that behaved correctly for a long time and became malevolent afterwards uses its long-term good reputation to harm the system.

3.4.1 The uncertainty of the truth

In traditional computational trust, we usually agree over a set of axioms and hypothesis. For instance, the "truth" is a notion that is common to all. A corrupted file is seen as corrupted by everybody. In spatial messaging however, the truth is context dependent. The truth becomes a subjective and temporal notion. Something that is true for one user is not necessarily true for the others. Something that is true at a certain time is not necessarily true later. We call this new notion the "uncertainty of the truth". If user *A* posts a tag saying "Dangerous path", user *B* only knows that user *A* finds this path dangerous. But *A* is perhaps just a tourist and the path is in no way dangerous for user *B*, which can be a confirmed mountain guide. Or this path was maybe dangerous because of the snow, which melt away by the time.

To our view, trust is not only a tool that can be used to exclude malevolent users from a given system. Trust is also a way of creating relationships between users that behave in a similar way. Like in real life, each user has its own definition of what the truth is. The aim is therefore to create trust relationships between people that share the same definition.

3.4.2 Contextual trust

The trust given to a user varies according with the context. A first example of context is risk. Trust and risk are closely related topics. There is no use to trust if there is no risk. More precisely, the amount of trust you need before undertaking an action is correlated to the risk and the costs of a negative outcome. You will probably accept more easily to lend 2 € to an acquaintance that you meet sometimes by chance in front of the coffee machine, than 2000 € to a close friend, even if the chance that you will never get back your 2 € is much higher. The way you update a trust value of someone else is also related to the costs involved. If your friend gives you back the 2000 € you lent him, you will probably qualify him as reliable, and perhaps lend him more money next time. However, if it the coffee machine acquaintance that returns you your 2 €, it is not sure that you will now trust him for a bigger amount. The opposite notion of risk is the benefit. Most of the time you undertake an action only if you get a benefit in return. If you see a tag mentioning an interesting exposition, you will compare the benefit (having fun) with the risk (loosing time) and then mentally compute the probability of a positive outcome according to your trust value in the tag's author.

A second example of context is the domain. You can trust someone for his knowledge in computer science, but have no trust at all in him for his tastes in cooking. When the domains are very different, we can simply decide to have one trust value per domain. In practice however, domains are not always that much different. There are often (more or less strong) links between them. If you know someone that excels in the domain of Java programming, you will trust him more easily to program you something in C++ than in repairing your car.
The third example of context is time and geography. If you know that a tag has been posted in a forest and using a GPS, we will be more tolerant about the precision of the positioning, since trees perturb heavily GPS devices. In the same way, you could decide to be more tolerant with a tag that has been posted by night, or a long time ago. Imagine that you see during summer time a tag that warns about a high risk of avalanches. Even if there is no snow anymore, it does not mean necessarily that the author was lying; it can also mean that the tag has been written six months ago.

The fourth example is additional information. Additional information is everything that is not included in the above text. For instance, if you see a tag signed by a trustworthy friend testifying very good food in a restaurant, you will not go there if you learned by another mean (newspaper...) that dogs are used in some meals. The tag can however remain true; it is a question of taste.

3.4.3 Updating trust values

A traditional way to store and update a trust value consists in counting the number of positive outcomes P, the number of negative outcomes N, and to define the current trust value as $T = P / (P + N)$. It is a simple model that fits very well to file sharing applications where a good file is simply considered as a positive outcome and a corrupted file as a negative one. In spatial messaging however, defining a positive and a negative outcome is more complicated. And since we have to deal with what we called previously the "uncertainty of the truth", we need to define a model that is specific for spatial messaging.

A model that can be used in case people are honest is one that uses data mining techniques in order to determine how reliable a given tag is, in a given situation for a given person. Data mining consists in picking up relevant information in large data sets. A good definition can be found at [70]. Basically, when you rate a tag, you increase the trust links with all the people that reviewed it in the same way, and decrease the trust links with all the people that rated it differently. While requesting tags, data mining algorithms are then able to determine how "close" you are with each reviewer according to the situations where you previously interacted with these people, and take this into account to determine how pertinent this tag is to you.

This model is however challenged when malevolent users take part in the system. For instance, an attack would consist in rating automatically and positively all new tags so that the next reviewers increases the malevolent user's trust value. And then this user will use its high value to post "reliable" false tags. A solution to this consists in increasing only the trust value of the author of a tag, since posting randomly interesting tags (if they are not "interesting", nobody will rate them positively) is almost impossible.

In applications where it is possible to scan all the tags, and rate them automatically, it seems easy to cheat the system. It is difficult in some cases to differentiate a normal behavior from a malevolent one. For instance, if you see a tag warning about a specific danger and you do not see this danger, you do not know if the author is a spammer (and you need to decrease his trust value) or if the danger simply disappeared (and then you should not decrease his trust value). We need to determine how much a trust value must be decreased when we rate negatively a tag, so that an honest user is not too much penalized, but so that a spammer can be excluded from the system in a reasonable delay. It means that even if the system is generic, it needs a high comprehension of the application domain in order to determine what are the right rules and parameters. For instance a rule will define how much we must decrease the trust value of someone that doesn't vote like us and a parameter will define what the minimum trust value is.

Like in the human world, trust varies not in the same way when it increases than when it decreases. Trust takes time be built, but can be destroyed very fast. And this non-linear way of handling trust is certainly necessary to protect ourselves. If you lent 10 times 2 € to someone that always paid you back, you will probably stop to trust him before 10 times when he stops refunding you. The reason is even more accentuated in a digital world where people can act in an automatic way, thus very fast. If we use our former P / (P + N) example, it is easy for a user to behave correctly (most probably in an automatic way) for a certain time, and then use its high trust value to subvert the system. A first idea consists in representing a trust value as a single value. A good behavior increases it, a bad behavior decreases it. But the maximal value is limited. It means that even if someone behaves very well for years, his trust value is not that high, and can quickly become negative in case of a big bad behavior, or a succession of a few bad behaviors. Another important point is that trust increases in a linear way but decreases exponentially. An exponential function varies very slowly at the beginning and then increases endlessly. Like in the human model, we accept to forgive seldom and small misbehaviors, but we break our trust relationships if you we face a big misbehavior or a succession a small misbehaviors.

3.5 Relation between trust and security

Are trust and security two completely different topics, which can be handled independently? The answer can be yes... ...or no!

Instinctively, as a first approach, we can consider that the two concepts are independent. Security deals with problem like integrity, authentication, non-repudiation and so on, in other words with what everyone would expect from a "secure" system. The security can be "measured" with metrics like the length of the key for a given algorithm or the quality of the algorithm itself. Trust, as interpreted by human communities, can be defined as the amount of risk that one is ready to take for a given action. In this sense trust and security are two different topics. In the example of e-banking, the security part will ensure that no third party is able to read or modify the exchanged messages, or to prevent any transaction from a denial of service attack.

The trust part concerns the trust I have in the bank. It is a good point that my money arrives safely in the bank, but it is useless if the bank is not honest and makes me loose all my money. But, again, there are clearly two independent concepts. Changing a security component of the bank should not modify the trust I have in it for managing my money. As time passes, the trust I have in the bank can increase even if the security components do not change. So, how to build a new trusted and secured application? We could first create a system without taking care about security and trust aspects, then we secure all the parts that need to be secured, and finally we add a trust mechanism. But this approach is wrong and dangerous. Among the most famous example, we could cite Internet. It was designed to be very open, and at its creation security aspects were not taken into account. Neither trust, of course. And everyone knows the current situation. We create more and more security tools, that we try to merge as well as possible with the rest of the system, but paradoxically the number of attacks and other annoyances, like spam, is increasing. Today, to surf the Internet, you need at least a firewall, an anti-virus, an anti-spywares, a spam filter, and be sure that all your software is up-to-date; tomorrow you will also need an anti-phishing, an anti-hoax, a scanner detector, a key-logger finder, and a cleaner that erases all sensitive information that can be found in your computer. Past experiences teach us that security is not a tool. Security is a process. And this process must start at the beginning of the design time.

That's why special care must be taken in order to avoid the same mistakes with the trust concept. We believe that trust is also a concept that must be included at the beginning, during design time. If we want to go beyond the simplistic e-banking former example, if we want to design a secured and trusted system, then trust and security have to be studied from the very beginning. It is important to note that in the e-banking example, the trust is not physically included in the system. The security concepts are implemented by algorithms, but the trust is only present in brains. It is only computed and managed by humans. But future applications will have to deal with trust in an automatic way. Trust must be interoperable, computable, and manageable by humans as well as by machines. Then, trust and security are in fact a single topic, or better said, a single process; a single process responsible to keep reliable the environment for which it was designed.

4 Model

4.1 Overview

Our model is very simple and can easily be applied to many situations. In short, users have a set of primitives to access a server, and this server runs a trust engine that allows a protected access to the tag database. We call our tags vTags (virtual tags) but use sometimes simply the term "tag" when there is no ambiguity. We can summarize our model through figure 1.

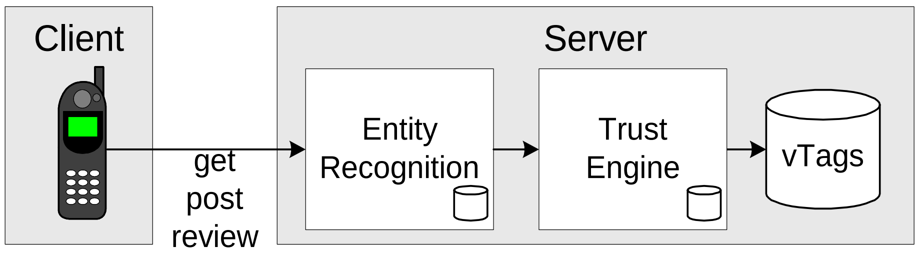


Figure 1. Our model

The client has three main primitives to interact with the server. He can *get* vTags, *post* vTags or *review* vTags. Getting vTags mean requesting all the tags that match a certain criteria. For instance, getting all the tags authored by Alice that are in a radius of 500 meters and that have been reviewed by trusty people. Posting a vTag means creating a new one. A user that posts a new tag is therefore the author of this tag. Reviewing a vTag consists in giving our advice about its content and/or giving a mark that reflects how much we agree with its content.

We see that there is no *delete* primitive. We will explain later why, but for now we just need to remember that a user that wants to delete a vTag will simply give a bad mark during a reviewing process.

Each request to the server goes through an entity recognition bloc. This bloc is the security component of this model and is responsible to check the identity of the requester and find the corresponding user ID. This ID, unique for each user, is then used to identify his owner everywhere in the model. For instance, this ID will be used to indicate who authored or reviewed a given vTag.

Once we are sure about the identity of the user, the request is transmitted to the trust engine which is then responsible to manage the vTags. For security reasons, a user cannot access directly the vTags. All the operations must be done through the trust engine. The trust engine prevents malevolent users to write false tags in the database, but acts also as a filter when answering a request in order to return only pertinent tags. To do that, it will use the trust links that the requester has with the author or the reviewers of the targeted tags.

4.2 Structure of a vTag

A vTag is structured into different standard fields. All the trust engines understand these fields, so even if the trust engine changes, the data stays. We could compare it to the web and the search engines: Even if we change the search engine, the content of the web remains the same.

We present in this section the standard fields of a vTag but we will see in the next section that a tag can be extended in order to be adapted to a particular situation or to a particular trust engine.

4.2.1 The ID field

An ID identifies uniquely a tag in a given environment. There are then several ways to access a given tag. A possible solution would be to match each ID with a specific URL. For instance, http://www.restorep.com/id=123 would request the tag whose ID is 123 in our restaurant reputation service. Using URLs allows us to see the tags in a web-browser (useful if someone does not have a specific application), bookmark them and send the URL to someone else. Another advantage is that an URL can be easily coded in different ways. It can for instance simply be written on an object, and people that want to see the tag attached to that object simply browse this URL. Or the URL can be transmitted by a RFID tag, so that the client application gets the tag when the user approaches the tagged object. In a similar way we can also use Bluetooth or wireless hot spots.

Another interesting way is to use visual tags, like described in Mobile Tag [29] (commercial) or in the MUSCLE European project [30] to signal the presence of a virtual tag. A visual tag is a small squared picture encoding an URL, and as we saw it previously an URL can match a given tag. This technology has the advantage of being very simply to use (just point the camera of your mobile phone to the visual tag), intriguing (people want to know what is hidden behind this strange image), and mobile (visual tags can be painted everywhere).

There are and there will be more different possibilities to present an URL. Using them to match them to an ID and therefore identify our vTags is a guaranty that future technological improvements will be easily integrated.

It must be clear that we do not need necessarily to know the ID of a tag in order to get it. To get all the tags around us, we simply need to give our current position. IDs just give us another way to access it, useful when the position is unknown.

4.2.2 The author field

Each tag includes a reference to its author. This allows a trust engine to compute the reliability of the tag. An author is represented by an author ID so that its real-life identity remains hidden.

In certain cases, to avoid a Sybil attack [31] an application can require that a single user owns only a single user ID (or at least a limited number of them). This can be achieved by different means. For instance we can use a reverse-charge SMS service like the one provided by [32], so that a single user cannot have more IDs than mobile phones. Or, if the user needs to stay anonymous even from the server, he can use blind signatures and a ZKP (Zero Knowledge Proof) algorithm as described in [33] in order to stay completely anonymous.

4.2.3 The position field

The position describes where the tag is. Unlike POIs, a vTag is not limited to a single point. A vTag can also cover an area. An example to express the coordinates is to use the WGS84 [75] standard (latitudes/longitudes/altitude), so that the system works worldwide. Taking the altitude into account allows us to draw virtual boxes in the air. It is therefore possible to show one tag to the people that are at the top of a tour, and a different tag to the people that are at ground level. An area can be very small: "here you are in my garden" or very big: "high risk of malaria in this region".

The position can be given by any technical mean, including GPS or GSM [76] Cell ID. The precision can therefore vary from centimeters (differential GPS) to kilometers (GSM Cell ID). The precision of the position is an optional field in a tag that is application specific. In this case, the precision could be given by the positioning device itself (GPS devices often know the precision of their measures) or given manually by the user. For instance a user that wants to tag an inaccessible object that is about, according to his estimation, 200 meters in front of him. This precision parameter can also be used in the future by technical means that compute distances. In the former example, we can imagine that the phone contains a built-in radar that computes the distance to the sighted object.

Tags can be mobile. We can imagine users that carry a tag with their profile in order to find neighboring users that share the same interests. Or a taxi company that wants to know in real time where the drivers are in order to send the closest one to a client. Theses tags are clearly meant to move and we expect that they send their new coordinates time to time. But we can also have a tag that was initially designed to be fixed, but that need to be placed somewhere else (for instance the tagged object has moved). In both cases, the "technical" operation consists in changing the coordinates of the tag. But then a history of the previous positions as well as the corresponding times must be kept. The reason is that the content or one of the reviews can be true only if the tag is at a specific position. It is the application that decides whether a tag can be moved, by whom (author, reviewers...), and what rights are given to the users (rating...) in each situation.

For some tags, it is not possible to give a position related to the Earth, expressed in latitude and longitude. In this case the tag can also be "attached" to a specific object (probably moving), so the position of the tag is simply the position of the object. The position is therefore done by a contextual description instead of a latitude/longitude coordinate. For instance, visual tags are painted on the boats participating to a regatta, and the spectators get information about the navigators by scanning the visual tags through the camera of their mobile phones. In this case, the vTag is attached to the boat, wherever it is.

4.2.4 The creation time field

In order to know the "freshness" of a tag, each new tag contains the date and time of its creation. We can for instance use the ISO 8601 [77] format (to be understood worldwide). The time stamp is set by the server (UTC [78]). The choice whether it is the server or the client that put the timestamp is important in cases where the delay between the writing of the tag and the reception of it by the server becomes significant. This can happen in a region where there is no network. A user records tags and they are stored on his device until a network connection is available. According to where the user is, a delay can be expressed in milliseconds as well as in days, weeks or months!

It is not a good idea to let the user to timestamp the tags. First, we cannot be sure that he is sending the right date, and second, since there can be various delays, we can have old tags that appear before new ones. This complicates significantly the handling of data by the trust engine, and can also create confusion by the users. For instance, a tag that should exist since a long time but which appears just now. These are the main reasons why we prefer to let the server to timestamp the tags. In this way, the creation time indicates in fact the time the tag has been made available.

4.2.5 The expiration time field

This field defines an expiration time for the tag. After this time, the tag is removed. Some tags report an observation that is true only for a certain amount of time, and sometime this amount of time can be defined by the author. For instance a mountain guide reporting a high risk of avalanches can clearly put an expiration time to its tag. It avoids spamming other users with outdated tags and helps the trust engine to evaluate how to adapt a trust value after a rating: If you disagree with the content of a tag but you are close to its expiration time, you will not decrease too much the trust value of the author since this tag is perhaps simply outdated.

Unlike the creation time field, the expiration field must be completed by the user, using for instance the ISO 8601 format and UTC time. We cannot just say how long a tag must be visible, and let the server to compute the expiration time. The reason is that we have no idea about how much time the tag will need to arrive to the server (we saw that it can be milliseconds or months), so the expiration information becomes useless. The tag will be visible only if it arrives on time. For instance if we send a tag today and put the expiration time in 10 days, then it will be visible only if the server gets it before 10 days.

4.2.6 The content field

The content of the tag is the materialization of the information that this tag provides. The content can for instance be coded in XHTML [79]. This language is sufficiently flexible and extendable to add specific fields for a given application, and sufficiently generic to be viewed by any web browser (for people that do not have an application for managing vTags). Doing so, we allow our vTags to contain the same type of content that we find on the Internet (text, pictures, videos...) and give them the possibility to link them though hyperlinks to other tags or to any web page.

4.2.7 The reviewers field

A user can agree or disagree with the content of a tag. A tag contains a reviewers list that is sorted in an inverse chronological order, since we want to give more importance to recent events and simply forget the ones that are too old. Each review contains the current time, the ID of the reviewer, the rating, and possibly some comments (same format than the content written by the author). The time of a review is the server time. The reviews must be independent from each other, and always comment the original content. It is not possible to review another review, so this must not be confused with a blog. This field is sometimes also called "history" in the rest of this work. To make the text easier to read, we prefer to speak about "reviewers" when the content of the reviews is important (for instance when the reviewers make comments about the tag), and about histories when the order of the reviews is more important than the content.

4.3 Extending vTags

A vTag can be extended in order to be adapted to a given situation. These extensions can be used by the trust engine or not. For instance, if we decide to add a heading so that only the users moving in the same direction as the author see the tag, then we can either decide that the trust engine takes it into account (it returns only the tag with a matching heading), or not (the trust engine returns all the tags and it is the client application that filters them). Figure 2 shows a vTags and its extensions.

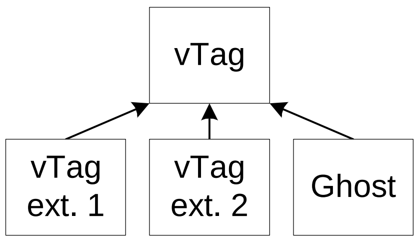


Figure 2. A vTag and its extensions

An extension provided by a given trust engine won't be visible to another trust engine. Only the standard fields will be understood by every trust engine. In comparison we could cite the web and several search engines: Each search engine creates its own index, but the content of the web (the standard part in our vTags) stays the same.

4.3.1 Ghosts

A ghost is the name we gave to a particular kind of extension. A ghost is used to mention that there where a traditional tag here, so it typically replaces a vTag that reached its expiration time. A ghost is active by default (it is visible), becomes inactive after a while (not visible anymore), but is reactivated if another vTag appears and disappears in a close neighborhood.

A ghost contains two additional fields, called "activation" and "delay". The "activation" field contains the time when the ghost has been activated for the last time. Note that at creation time this field is set to the current time, so a new ghost is immediately active. A ghost is sent to the client only if it is active. The "delay" field indicates for how long the ghost must stay active. There is a starting default value but this value can increase with the time. A ghost that stays inactive for a given delay is deleted.

When a new ghost is created, it first checks in its neighborhood if there are other ghosts and gets the biggest active delay of them. If no neighboring ghost is active, this delay is increased. Finally all the neighbors as well as the new ghost are set with this new value and activated.

The fact that the ghosts are related between them allows painting virtual zones where an event is likely to happen. For instance, if we sent a vTag (which then becomes a ghost) each time we see a lion, then little by little we will create a network of ghosts so that when one is activated (someone saw the lion), the other places where the lion already came at least ones are activated as well. We can use the same concept for finding good haunting places.

A completely different example can be applied to mobile speed cameras. When a tag mentioning a mobile speed camera disappears, it is replaced by a ghost. And since speed controls are often made at the same places, these ghosts allow drawing maps indicating the probability of crossing a mobile speed camera in a given area. This speed cameras example can also be used to illustrate why it is important to increase the active delay value only if no neighboring ghost is active. Consider the following scenario: A mobile speed camera disappears, so the corresponding tag is transformed into a ghost. This ghost stays active for 10 days (the values here are only used to better illustrate the scenario), which means that other drivers will be warned about the risk of crossing a speed camera here during this delay. After 10 days, the ghost becomes inactive. If later a new ghost appears in the neighborhood, it means that we are perhaps in a zone where there are often speed controls. So it is worth to increase the active delay of all the neighboring ghosts, let's say 30 days. And so on until a given limit, for instance 365 days. However, we saw previously that we increase the active delay only if no ghost is active in the neighborhood. This allows to better taking care of events that appear frequently but only for a certain time. In our speed camera example, consider a roadwork where the workers are protected by a mobile speed camera. If the camera appears everyday, and therefore the ghost is re-activated everyday as well, we do not want that the active delay increases too much. We want that the ghosts become inactive as fast as possible as soon as the roadwork is over.

4.3.2 Custom extensions

Extensions are not limited to ghosts. An extension can be added in order to specialize a vTag for a given situation or for a given trust engine. We give here a little example to better show what kind of extension can be made. Consider that the position of a tag is given by a latitude and a longitude and that the tag is visible only if the user is in a radius of 500 meters. A possible extension would be to allow any area instead of just a circle centered at the current position. We could then define a structure (in this example we use XML) and have:

```
<visibility>
  <rectangle>
    <n_w_lat>46.33042</n_w_lat>
    <n_w_lon>6.34344</n_w_lon>
    <s_e_lat>46.33020</s_e_lat>
    <s_e_lon>6.34366</s_e_lon>
  </rectangle>
  <oval>
    <n_w_lat>46.33050</n_w_lat>
    <n_w_lon>6.34350</n_w_lon>
    <s_e_lat>46.33012</s_e_lat>
    <s_e_lon>6.34360</s_e_lon>
  </oval>
</visibility>
```

"n_w" means North-West, or the top left corner (of a rectangle), "s_e" means South-East, or bottom right corner. For ovals, this coordinates correspond to an imaginary rectangle that just contains the oval. The example above describes a visibility zone that looks like the one in figure 3.

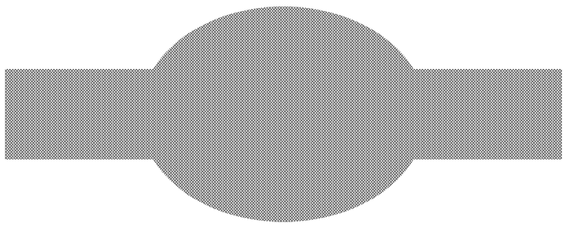


Figure 3. A personalized visibility zone.

A combination of basic shapes allows to defining every random visibility zone. As already mentioned previously, this extension can be handled by the trust engine (it then returns the tag only if the user is in the visibility zone) or not (the trust engine returns anyway the tag and it is the application that checks whether the tag must be shown).

4.4 The trust engine

A trust engine is responsible of managing the vTags, to keep the trust links between the users, to exclude malevolent users and to act as a filter in order to return only tags that are relevant for the requester. Each trust engine keeps therefore some data that are used, among other things, to compute the trustworthiness of a tag. Figure 4 shows a trust engine.

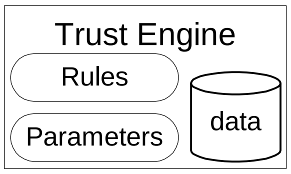


Figure 4. A trust engine

A trust engine must be as generic as possible in order to be reused. However, the way trust information is handled is very application dependent. This is why a trust engine can be customized through rules and parameters. There are two sets of rules. The first defines whether a given tag must be returned. For instance, returning tags only if at least two other trusty users reviewed them. The second set defines how the trust values must evolve during a review. For instance, increasing the trust value of the author if we agree with the content of the tag, and decreasing the author's trust if we don't.

The parameters are only values. For instance a parameter could define how many users must disagree with the content of a tag before starting a tag removal process. The parameters can be used by the rules or not.

We will now see the different actions that can be done on vTags.

4.4.1 Creating a vTag

A user that wants to post a new vTag sends a request to the server. He adds to his request the information about the tags, like the position and the content, and provides also his identification data, like a username and a password. The entity recognition bloc replaces the identification data by the corresponding user ID and transmits the data to the trust engine, which creates the corresponding tag and stores it in the database.

4.4.2 Requesting vTags

There are several ways to search specific vTags. The first consists in using contextual data, like the position or the time the tag has been created. For instance, asking all the tags around my current position that have been created during the last 24 hours. The second solution consists in using keywords. The content of a tag can be structured according to the needs of the application; we can therefore have for instance a "title" section or a "keyword" section. We can then ask to search all the tags that contain the given keywords, like it is done in Flickr [35], a service that allows to geocode and share photos with others. Note that Flickr uses the term "tag" to define what we call "keyword", which can be a bit confusing. A third solution consists in using the title of the tag. For instance, if Alice drew a virtual path using tags and called them (title) "waypoint 1", "waypoint 2" and so on, an interested user can therefore ask his system to show all the tags authored by Alice where the title starts with "waypoint". These are only basic examples of searches. We can also imagine that searches are done in the whole tag through regular expressions [36], which would allow to find, for instance "all the tags around me, authored or reviewed by Alice during the last 12 hours, that does not contain 'waypoint' in the title, with either 'cat' or 'dog' in the keywords".

In order to save bandwidth and computer power, all the operations are always done on server side. For instance, an application could by default download only the titles of all the tags around the user's current position, and download the full tag only if the user expressly requests it.

4.4.3 Reviewing a vTag

Reviewing a vTag consists in adding some information to it. Reviewing consists in rating the tag, in commenting it, or both. Rating a tag consists in giving a number to indicate how much we agree with this tag. Typically we use a value in [0..1] so that 0 means a full disagreement, 0.5 a neutral opinion, and 1 a full agreement. Applications can either stick to these three values, or use all the floating point numbers in-between. The rating is a precious indication for the trust engine. It allows to return to a requester only relevant tags (tags where the requester's friends agree), to remove outdated tags (nobody agrees anymore), and to exclude malevolent users (nobody never agreed with his tags).

Commenting a tag, which can be done alone or in combination with a rating, consists in adding a comment related to the original post. It can be used to justify a rating or simply to add complementary information.

4.4.4 Revoking, denying and deleting a vTag

To avoid malevolent acts, only the trust engine is able to delete a vTag. This primitive (deleting a tag) actually doesn't exist for the end user. The only thing the user can do it either to revoke a tag or to deny it.

Revoking a tag is a primitive reserved for an author. It consists in adding information that says that the author doesn't agree anymore with his own tag, for instance because it is outdated. A trust engine can be configured to delete directly revoked tag, but in practice this can be dangerous; malevolent users could use it in order to artificially increase their trust value, as it will be explained later in this document.

Denying a tag consists simply in giving a bad mark while rating a tag. It is the opposite of confirming a tag, which consists in giving a good mark. A user wanting to remove a tag written by someone else will deny it. This allows the denier's friends (the people that have a high trust value in him) not to be disturbed by this tag anymore. And then, if a tag has many bad ratings, the trust engine can decide to launch the process of deleting the tag.

Deleting a tag is an operation reserved to the trust engine that is divided in two parts. The first consists in launching the process. The tag receives what we call a request-to-delete order. For instance, a trust engine could decide that a tag receives a request-to-delete order when two successive users denied it. But it would be too dangerous to remove the tag directly, since then two colluding malevolent users would be able to remove all the tags. Instead, the tag remains still present for a while so that other users can rate it and then help the trust engine to exclude malevolent users. The second part in the tag deleting process consists in removing it definitively. If no operation cancels the request-to-delete order (for instance a positive rating) in a given delay, the tag is deleted by the trust engine. We understand now that giving a request-to-delete order technically consists simply in setting an attribute with the current time, and if this value is not reset after a while, the tag is deleted.

4.5 Specific points

This section is a discussion on some specific points that concern both 4.2 - "Structure of a vTag" and 4.4 - "The trust engine".

4.5.1 Shadow areas

We call shadow area a zone where the user is unable to know its position, like in an urban canyon or a tunnel if he uses a GPS. There is no generic and efficient way to handle this issue. Especially since it is actually only a technical problem. In the future, GPS devices will improve and other positioning systems will appear. And a shadow area is not necessarily one for another user, equipped with a more sophisticated positioning device. It is therefore the application that decides how to handle this problem. For instance, visitors of a national park get vTags thanks to the position given by their GPS while outdoor, and thanks to visual tags while inside grottos.

4.5.2 Tracking

There are two ways of doing tracking. The first has already been discussed previously and consisted in modifying the position of the tag. This can be used if we are interested to know where a person or an object is, at present or some times ago. The second way consists in sending a new tag each given time interval, and so draw a virtual path. For instance the GeoSkating [37] application built over the GeoTracing [38] generic framework allows skaters to draw maps, where the roads are colored according to the quality of the surface.

It is possible to create virtual paths with successions of vTags. We can decide that these tags can only be rated by the author (in order to delete them), and that the content is limited to a number corresponding to a road surface quality. And in some countries where maps are inexistent or forbidden, this will allow to create them in a collaborative way.

A particular way of doing tracking is passive recording. We do not need necessarily a human interaction to send tags. For instance, we can record every second all the information sent by the GPS (like the speed, the altitude...) during a ski trip, and then analyze our journey on Google Earth [72]. But we can also collect data through other sensors (for instance the concentration of pollens, the signal strength of a mobile phone operator, the temperature...) and fill a geo-referenced database. If we have enough data, it becomes possible to interpolate the values for every position. We can therefore draw useful maps, like one indicating the concentration of pollens, or one that show where the network coverage of a phone operator should be improved.

4.5.3 Simultaneous updating

This happens when two users are rating a tag simultaneously. For the system this is not a problem since the time of the rating is the time given by the server when it gets the information. In other words, there is no simultaneous updating for the server. At the client side, it is not a problem as well. Remember that a review is only about the original text (we can not review a former review). Since the reviews are therefore completely independent from each others, the order they arrive is irrelevant.

4.5.4 Reverse trust updating

Reverse trust updating is an automatic process that consists in rating the people that are rating you. For instance, an author can decide to rate positively all the users that rate his tag positively as well, since they seem to share the same opinion. There are two main advantages in doing reverse trust updating. The first is to speed up the creation of trust relationships. And the second, the most important, is to motivate people in publishing virtual tags. Otherwise the only motivation for an author is to update his reputation. But since this value is limited (to avoid future malevolent acts), there is no interest for an author to post lots of tags. However, if this allows him to build bidirectional trust relationships, so that the quality of the information he will get in the future improves, then the author has a clear motivation in publishing new tags.

However reverse trust updating can be easily exploited by malevolent users. The easiest way consists in scanning the system for new tags and to rate them automatically and positively, so that the authors increase the trust values of these malevolent users. To scope with that, we found mainly two solutions. The first is to hide the information and not allow system scans. For instance, a user that rate tags in different remote places within a short amount of time is suspicious. But again, defining what is suspicious and what is not is directly related to the application. This is especially true if we allow also using the system without being in the field. The second consists in posting false tags in inaccessible places. If someone rates all the tags in an automatic way, he will soon or late fall in such a pitfall. And then the system can simply exclude this user from the system. Again, posting pitfalls is an operation that is application specific and that probably needs human interaction.

In short, reverse trust updating brings some advantages (motivation to post new tags, and speeding up the process of creating trust relationships), but needs to be used with greatest care since malevolent users can use this opportunity to increase easily their own trust value in order to subvert the system.

4.5.5 Web of trust

The notion of Web of trust has been defined by Zimmerman [39]. It says that if *A* trusts *B* and *B* trusts *C*, then *A* trusts *C*. The weight of a trust relation decreases with the number of levels. For instance, if *A*'s trust in *B* is 0.9 (out of 1), and *B*'s trust in *C* is 0.5, then *A*'s trust in *C* could be 0.9 * 0.5 * *k*, *k* being a constant in the interval]0..1[. A web of trust allows you to trust people to trust other people.

This notion is very useful in spatial messaging for big communities. Indeed, when you join a community, you do know the others and you are not able to trust them either. You need actually to interact with every of the users in order to know if you will be able to trust them next time. In a web of trust system, you just need to make a few friends. When you later face a tag signed by an unknown author, you will be able to ask your friends if this author is reputable. And your friends will ask their own friends, and so on up to a certain level.

To judge the trustworthiness of a tag, we combine the local trust value (our own opinion of the author, computed during past interactions) with the trust values of our friends (who also asked their own friends...). Each of these two values is weighted according to the application domain. For instance we could decide to take 20% of the local trust value and 80% of the value given by our friends. It is important to note that we always ask our friends for advice, even if we have a local trust value for the author. This allows us to still trust someone even if we gave him previously a bad rating, for instance if we saw a tag warning about a danger that disappeared. But this avoids us also to trust "old friends" that became malevolent since the last interactions we had with them.

4.5.6 Passive behavior

A passive behavior is when a user observes something that deserves to be tagged (like a rock on the road), but he does not report it. It can have its importance for the trust engine. Imagine you know that a given user drove on a specific road a few minutes ago. You could think that he will tag any danger, so if there is not tag, it means that there is probably no danger as well. But perhaps there is a danger and the driver could not tag this object, simply because he is alone in his car and it is thus not save to handle his mobile device while driving. Or he just wanted not to tag this object for any other reason. Since there is missing information (why didn't he reported a given fact?), it is very difficult to know how the trust value of a passive user has to be updated. And ignoring a passive user is still losing some information.

We prefer therefore a different approach. The idea is simply to post also tags when there is nothing special, rightly to signal that there is nothing special. Remember that a tag can occupy any area. So a driver can also post a tag that covers exactly his path and that informs that there is no danger. Any following driver will therefore know that while he is inside the tag area, the road is safe. This simplifies a lot the way the trust values are handled, since we can then simply ignore people that do not tag.

4.6 The entity recognition

Each request to the server goes through an entity recognition bloc. This bloc is the security component of this model and is responsible to check the identity of the requester and find the corresponding user ID. This ID, unique for each user, is then used to identify his owner everywhere in the model. For instance, this ID will be used to indicate who authored or reviewed a vTag. Behind the entity recognition bloc we are in a secured environment. It doesn't mean that it is impossible to harm the system, but it means that every action is related to a given ID, and this ID is related to a given person. Secured doesn't mean trusted: Even if now we are sure that a given tag has been posted by John, it doesn't mean that we can trust John.

There are many ways to achieve an entity recognition process, as it has been deeply studied in Jean-Marc Seigneur's PhD thesis [71]. The challenges are, among others:

- Make sure that a single user owns only one, or a maximum number of different IDs. If the same user can create as many identities as he wants he can launch a Sybil attack [48].
- Protect the privacy of the users, for instance allowing a user to stay anonymous.

An entity recognition bloc is also responsible to create a unique ID for a given user. It must therefore be able to identify uniquely a given person, and be sure that the same person cannot have as many IDs as he wants.

4.7 Attributes

We define some attributes to categorize our trust engines and our vTags. Since a given trust engine is related to the tags it is handling, we can often use the same attributes. For instance, we can speak about a single-fact tag as well as about a single-fact trust engine. Cases where an attribute can only be used either with tags or with trust engines are specifically mentioned.

4.7.1 Single-fact attribute

We define that a vTag is single-fact if its content gives a single piece of information where all the users should see it either as true or false (everybody should have the same opinion, it is an objective content). For instance, a tag that indicates that there is a restaurant at this place is either true or not. We could not however have a tag that indicates that the restaurant at this position is very good, since this appreciation is subjective. The trust engine is responsible to remove out-dated tags and exclude spammers (people that post false tags) and malevolent deniers (people that wants to delete true tags). The difficulty is to find out if someone that denies a tag is a contributor (the tagged object really disappeared), or a malevolent denier. We have the same problem for someone that posts a tag, where the job is to find out if it is a useful contribution or spam.

4.7.2 Mutli-fact attribute

A mutli-fact vTag is ones that give several pieces of information where all the users should see it either as true or false (everybody should have the same opinion). For instance, a tag that indicates a restaurant gives also its phone number, the kind of food and the opening hours. In a single-fact trust engine we would have a problem if the opening hours change, since then the whole tag wouldn't be considered as true anymore. In a multi-fact tag however, a user can just disagree with the opening-hours.

There are good similarities between multi-fact tags and some geo-referenced pages in the free Wikipedia encyclopedia [40]. First of all, an encyclopedia is meant to present facts and not the personal opinion of the author. Then, as we can observe it in pages that describe a place or an object attached to a specific position (like a monument), there are more and more geographic coordinates attached to it. A user can then click on these coordinates in order to see it on a map. Google Earth [72] included a layer with these geo-referenced pages, which means that we can easily imagine a mobile user that sees Wikipedia pages according to its current position. In this sense, these pages can be considered as a kind of virtual tags.

4.7.3 Single-opinion attribute

A single-opinion vTag gives the personal opinion of a user for a specific place, it is therefore a subjective tag. For instance, a user can post a tag saying that the restaurant at this position is very good and therefore highly recommended. However, a second user should not consider him as a spammer if he disliked the food. The role of the trust engine is therefore not to exclude malevolent peers, but only to create links between people that share the same opinion. A requester that wants to go to a restaurant can for instance search all the opinion tags posted by friends, or friends of friends, as well as by trustworthy people. After his meal, he can update the trust values of the authors of the tags. Such a system can easily be combined with existing POIs provided by a single entity. For instance, you can search in your navigation system all the neighboring restaurants (which should only gives you facts) and then check if there are opinion tags posted at the same position. The difficult part here will be to create categories and deal with the links between the categories. You can for instance trust Martin for his recommendations for restaurants but not for other topics. And there are links between topics. If you trust someone for his knowledge in programming Java and C++, then you will more easily trust him for his knowledge in Java Micro Edition than in cooking.

4.7.4 Multi-opinion attribute

In a multi-opinion vTag a user can give his opinion about different topics. For instance, he can say that he found the food very good but the choice of wines is too limited. Someone that does not drink any alcohol can therefore use his opinion about the food, and trust it, and do not take care about the limited choice of wines. Again, a trust engine will have to face the same difficulties, namely to deal with different categories and with the links between the different categories.

4.7.5 Binary attribute

A binary vTag is one that has no content or one where the content is predefined and unchangeable. It is therefore a specific single-fact tag where the content is always the same. This kind of tags seems to be very limited but is actually quite useful in practical situations. There are lots of application domains where a binary tag is sufficient. For instance, we can use binary tags to signal speed cameras on a road, or dangers, or accident. We can report every event or object that appears in different place. Another example could be to post a virtual tag at each position where you catch a fish, and then little by little draw a map of the best fishing places.

Another advantage of binary tags is that there are easy to post. Since there is no content to write, and that the other information like the position, the time, the current speed or the current heading can be computed in an automatic way, we can imagine that applications would need only a single pressure on a button in order to create a new tag. In practice, this is very important. Imagine a driver that wants to report dangers on a road; it wouldn't be safe if he had to write a message each time he wants to post a new tag.

It is easy to post binary tags, but it is also easy to warn the user about the presence of one. Since there is not content, we can use a single alarm to mention to the user that he is approaching a tag. Drawing a red point at each tag position on a map gives information about the repartition of the tagged objects or events, like for instance the best places for fishing.

4.7.6 Color attribute

A colored vTag is a single-fact one, but with a limited number of different contents. We define that a given content corresponds to a given color. This definition is purely arbitrary and has been taken from the practice, where we observe that tags are generally represented by different colors according to their content. Colored tags can be used in situations where the number of different objects or events to tag is limited, like for instance in the case of an application that allows to a driver to signal speed cameras, dangers and accidents. A binary tag is actually a mono-colored tag.

A trust engine for colored tags can work either like one designed for binary tags, or like one designed for single-fact ones. If the different colors are unrelated, we provide a trust engine for each color. We have therefore a set of trust engines designed for binary tags. But if the different tag contents are related, then the trust engine needs to take the content of the tag into consideration. In this case, the trust engine will work like a simplified trust engine for single-fact tags.

4.7.7 Position-dependent attribute

When we talk about position-dependent attributes, it means that we are interested in how two or more tags are related according to their position. For instance if a tag with the same content appears and disappears in a given neighborhood, what does it mean? Let's imagine users that add tags when they observe a given event. With enough tags, it becomes little by little possible to draw areas where this event appears. The position-dependent attribute can only be used by a trust engine. There is no sense to talk about position-dependent tags, since they are not different from the others. The difference is the relation between them, and this relation is handled by the trust engine.

We do not talk here about the precision of the positioning, nor about technical issues like a shadow area (a zone where the positioning device is unable to get a position due to external conditions). The reason is that these issues can actually be taken into account in the multi-fact attribute. For instance, we can decide that all the tags must be posted with a precision of 10 meters. If this fact is not respected, then the tag cannot be trusted for the accuracy of its position. But the other facts, like the useful information contained in the tag, can still be trusted.

4.8 Nomenclature

This section defines some of the most important terms that we use throughout this work. The complete list, sorted in alphabetic order, can be found at 10 - "Glossary".

- **Voting for a vTag**: Giving to a vTag a mark that reflects how much we agree with the content of this vTag.
- **Positive vote**: A vote that indicates that we agree with the content of the vTag.
- **Negative vote**: A vote that indicates that we do not agree with the content of the vTag.
- **Rating a vTag**: Same as voting for a vTag.
- **Positive rating**: Same as positive vote.
- **Negative rating**: Same as negative vote.
- **Confirming a vTag**: Consists in giving a positive vote to a vTag, meaning that we agree with its content.
- **Confirmer**: Someone who confirms a vTag.
- **Denying a vTag**: Consists in giving a negative vote to a vTag, meaning that we do not agree with its content. In some case, it means that we want to erase this vTag.
- **Denier**: Someone who denies a vTag.
- **Commenting a vTag**: Consists in adding a human-readable part to the vTag, either to complete the initial information, or to justify the corresponding rating, or booth.

- **Reviewing a vTag**: Consists in rating a vTag, in commenting a vTag, or both.
- **Rating a user**: Means updating his trust value.
- **Trustor**: The person who trusts. A trustor trusts a trustee.
- **Trustee**: The person that is trusted. A trustor trusts a trustee.
- **Local trust value**: Direct trust that one user has in another user, based only on former direct interactions between the two users.
- **Friend**: Someone in whom we have a local trust value higher than 0.
- **Combined trust value**: The combined trust value is the trust that one user has in another user, while combining its local trust value with the combined trust value given by his friends. It is a recursive function, meaning that a given friend will actually return a combination between his local trust value and the values returned by his own friends, and so on up to a certain level. The combined trust value is the one that is used in most case in human communities, where people use their own experience as well as recommendations given by friends in order to determine how trusty a given target is.
- **Global trust value**: The sum of all the local trust values for a given user. It allows to find malevolent users and to exclude them. This must not be confused with the mean trust value.
- **Mean trust value**: The average of all the local trust values for a given user. Despite the appearances, this value cannot be used to determine how reliable a user is. The reason is that this value changes if we add new users at the other side of the Earth, even if they do not interact with the existing community. To determine how reliable a user globally is, we use the global trust value.
- **Reputation**: The reputation of a user is proportional of the trust that the other users have in him. If *A* increases the trust he has for *B*, then *B* increases his reputation in the eyes of *A*. The term "reputation" is mainly used when the term "trust" is ambiguous, like for instance in "the trust value of *A*" which can design the trustworthiness of *A* or the trust value that *A* has for another user.
- **Local reputation**: The reputation that one user has in the eyes of another single user. See Reputation.
- **Combined reputation**: The reputation that one user has in the eyes of another user and his friends. See Reputation.
- **Global reputation**: The sum of the local trust values that all the other users have for the targeted user. See Reputation.
- **Mean reputation**: The average of the local trust values that all the other users have for the targeted user. See Reputation.

5 First steps towards virtual tags

Trusting virtual tags is clearly a new and promising concept. To be sure that today's technology will allow us to develop test applications in order to prove our ideas, some preliminary work have been realized with the help of bachelor and master students.

5.1 Bluetooth GPS API

This work has been done by Christophe Praplan and Stéphane Velen. The problem was to couple a mobile device with a positioning system. They created an API that runs on Java-compatible phones and that is able to get information provided by an external Bluetooth GPS. A test application provided with the API lets the users to obtain some geo-related information (position, altitude, heading, speed...) but allows also recording tracks (to see them later on a map for instance), compute distances, and configure the GPS device via the Bluetooth protocol. This library has been used by other master students and can be found at [21].

5.2 Dispatch-Locator

It is quite difficult to manage first-aid people in a crowded place. To solve this issue, Joëlle Levi and Stéphane Krattinger developed Dispatch-Locator, an application to dispatch emergency people efficiently. The first-aid people from the city of Geneva where voluntary to test the system. It has been deployed during "les fêtes de Genève". A coordinator sees on the main screen of its desktop application all the users on a map. He can send messages to them, or a new position to reach. The users own a mobile phone coupled to an external Bluetooth GPS. They send automatically to the coordinator from time to time their new position. When they get an order to reach a new position, they receive the description of the mission and are guided to this specific point with an arrow that heads to the destination and a counter that evaluate the remaining distance. Figure 1 shows a screenshot of the coordinator application and a screenshot of the mobile application, where the user is directed towards his new mission.

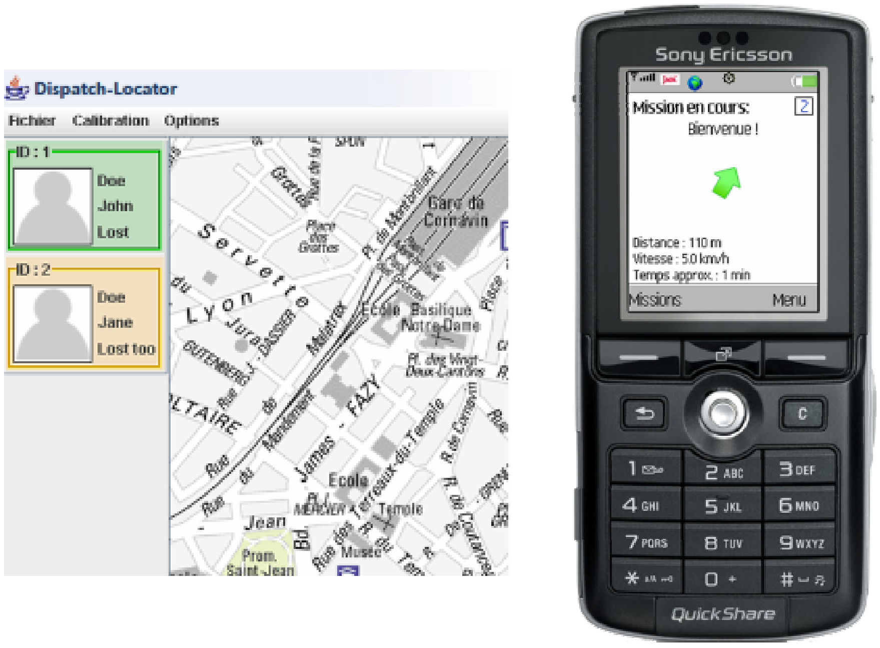


Figure 1. The coordinator and the mobile application

5.3 ViewMap

This was the first attempt in our research group to add maps directly on the mobile phones. The idea was to get bitmap maps from different sources, calibrate them, and let the client application choose the best map according to the requested position and zoom. This work showed however that mobile phones are currently not ready (too slow) to deal efficiently with huge amount of data. The authors, Ludovic Vollmer and Quentin Miché-Santoro, concluded that it would be much better to use vector maps, even if the precision of their details are far behind bitmap maps. More information can be found on their website [22].

5.4 Mobil-i

In collaboration with the public transportations of Geneva (TPG), Joseph Duteil developed Mobil-i, an application that runs on a mobile phone coupled to a Bluetooth GPS. This application shows a map of the TPG network as well as the user's current position. The user can find the closest bus stop thanks to his GPS and check the timetable of the next departures thanks to a connection to a central server. Once he selected a departure, the application guides him to the station and shows in real time the remaining time in order to catch the bus. More information can be found at [23].

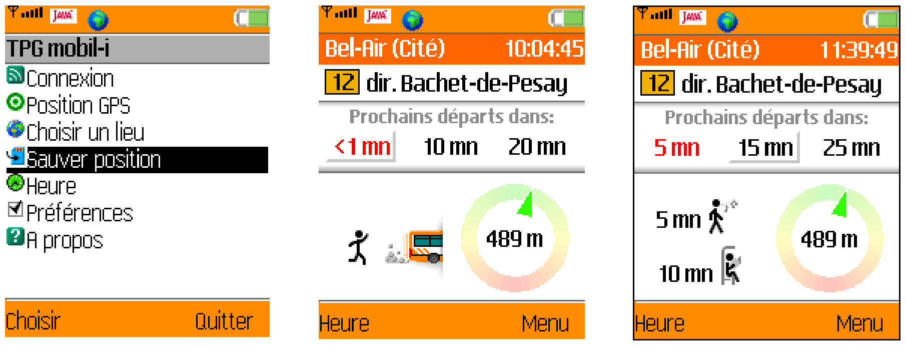


Figure 2. Screenshots of Mobil-i

Figure 2 presents 3 screenshots. The first shows the main menu of the application and the two next indicates the next departures for the selected bus stop as well as how to go there.

5.5 Deployment of J2ME applications

"Write once, run anywhere" is perhaps the most well known saying of Sun, the inventors of Java. It seems however that it doesn't remain true when we deal with the mobile version of their language, J2ME. The reason is quite simple. In the standard edition, it is Sun itself that provides a virtual machine for each platform. But in the micro edition (the one used by most mobile phones), it is the manufacturer that implements the requirements of Sun. We observe therefore some differences due to omissions (they "forget" to implement some functionalities), modifications (they provide a different API for the same functionality), or strategic decisions (some API can only be accessed by applications that are digitally signed by the manufacturer). This explains why we find different versions for the same application on most download sites. Two master students, Dejan Munjin and Hikari Watanabe, studied these differences and wrote a full report attended to newcomers in the world of mobile phones developers. This document can be found at [24].

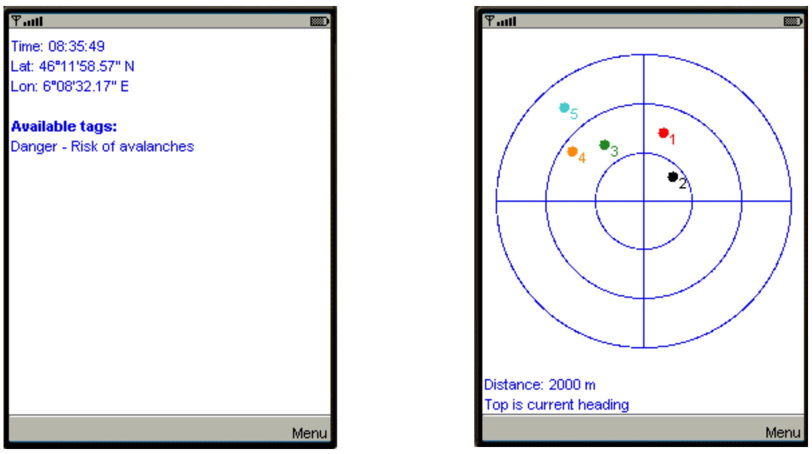
5.6 GeoVTag TA

GeoVTag is our future generic platform (at this time only a part of the client side was developed) containing trust engines and geo-related tools. To test the basic functionalities, and to test in the field how easy and practical it is to deal with virtual tags, we developed the GeoVTag Test Application (GeoVTag TA). Some functionalities:

- Get and post virtual tags.

- Automatically post tags in order to draw a virtual path.
- Record tracks with time, position, altitude, heading, speed and other geo-related information in order to show them on a map.
- Search virtual tags according to a search pattern and a given radius.
- Displaying of the neighboring tags in a graphical screen.
- Displaying of the direction and the remaining distance to a given tag.

GeoVTag TA was the first application for mobile phones that has been developed in the scope of this PhD and was since its beginning designed like an API. It means that all the basic functionalities that are not present in J2ME (Java Micro Edition), like printing a menu, reading and writing files, saving parameters or other geo-related tools have been put in different APIs. These APIs have been used by the other projects described in this chapter and will be the first stone of our future generic framework. More information about this application can be found at [25].


Time: 08:35:49
Lat: 46°11'58.57" N
Lon: 6°08'32.17" E

Available tags:
Danger - Risk of avalanches

Distance: 2000 m
Top is current heading

Figure 3. Screenshots of GeoVTag TA

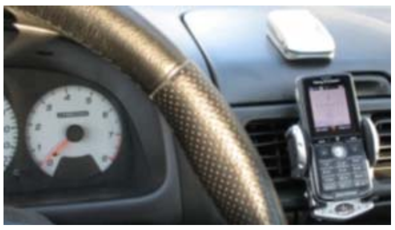
Figure 3 shows two screenshots. In the first we see all the available tags at the current position, and in the second we see all the tags in a given radius.

5.7 FoxyTag

FoxyTag is a collaborative system to signal speed cameras on mobile phones. The idea consists in posting virtual tags close to speed cameras in order to warn the other drivers. These users will then get an alarm when they are closer than 15 seconds to a critical point, and a red point locating the speed camera appears on their screen. You can signal a fixed speed camera by pressing the key "1" of your mobile phone, a mobile one by pressing key "2" and that a camera disappeared (you get an alarm but you do not see any speed camera) by pressing "0". You are also invited to signal speed cameras that have already been tagged; by confirming their presence, you create trust links with other users and get more reliable information. The system excludes automatically users that do not vote "like the others". Roughly speaking, the more you participate, the more the information you get is reliable. FoxyTag is a "collaborative" system. Tags posted by FoxyTag are directional. So, tags posted for users driving in the opposite direction won't be signalized to you.

FoxyTag motivates neither speeding nor any other risky behavior, but allows the driver to concentrate on the road instead of having is eyes fixed on the speedometer, by fear of being flashed. We observe that drivers tend to brake suddenly when they see a speed camera (even if they are not speeding), which can provoke traffic jams or even accidents. FoxyTag signals in advance the presence of speed cameras, so that the driver has enough time to check its speed and adapt it if necessary.

FoxyTag has been chosen to test our first trust engines since it was very easy to find volunteers to download the application and buy a Bluetooth GPS receiver. More information about FoxyTag can be found at [26].



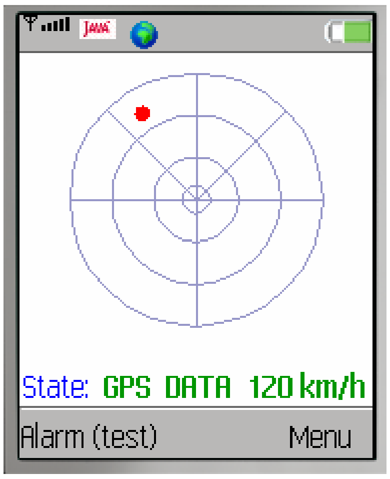


Figure 4. FoxyTag in action

Figure 4 shows a picture of a mobile phone connected to an external Bluetooth GPS and a screenshot of the application where the red point represents a speed camera.

5.8 FoxyNav

FoxyNav is a project done in collaboration with arx iT [27], a society specialized in GIS, and the SNG (Société Nautique de Genève) [28], which became famous since one of its boat, Alinghi, won the two last America's cup.

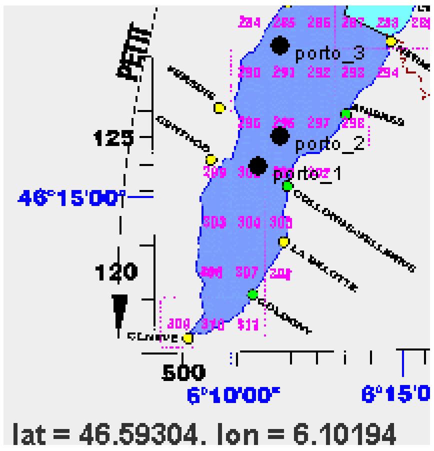
There are 34 rescue teams around Lake Leman. However, during big events like the Bol d'Or (about 600 participants), an additional team is used to coordinate the rescue operations. With their 22 boats, they are responsible to supervise the all race and contact the nearest rescue team in case of an emergency.

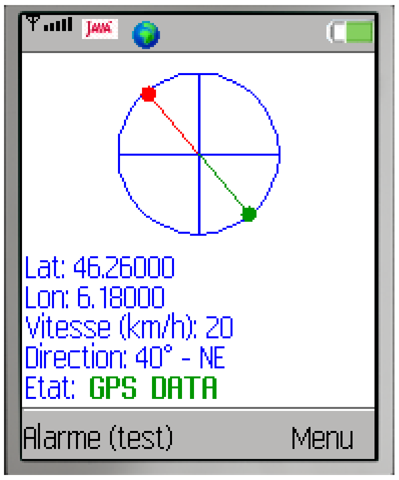
Porto Central is the name of the coordination centre of these 22 supervision boats. Formerly, they communicated only via radio messages and the positioning was done through paper maps. Lake Leman is divided in 311 squares and the precision is therefore limited to them. And this is in the best case, if we suppose that each boat knows in what square it is (in practice it is seldom the case). Porto Central estimates that in case of an emergency only 35% of the navigators where able to transmit their GPS coordinates.

We developed therefore FoxyNav, a solution that provides a desktop computer application that shows all the Portos (name given to the Porto Central boats) on a map. The coordinator can then move them with his mouse to specific places. On the boat, the user receives the new location to attend on his mobile phone. Coupled to a GPS device, the mobile phone continuously shows the direction to take in order to attend the new position. The mobile phone is also responsible to send each given time interval its position in order to update the coordinator's map. The system is divided in three parts:

- The server. All the data (positions of the boats, current orders to move a boat...) are kept on a server machine that is protected against blackouts and that is able to restart automatically in case of failure.
- The coordinator. The coordinator is an application designed to run on a desktop computer or a laptop. It shows all the boats on a map and allows to the user to move boats and send them new missions, like a new position to reach. Actually, the coordinator is only a graphical user interface that allows interacting with the server. It means that even is the coordinator shuts down or looses its Internet connection (which can happen quite often in the middle of a lake), the boats that run the mobile application are not affected. The coordinator can therefore also restart on another computer.
- The mobile application. Each navigator runs on his mobile phone coupled to a GPS the mobile application. Its connects from time to time to the server in order to update its current position and checks at the same time if there are new messages for it (for instance a new mission or an order to move to a new position).

Figure 5 shows two screenshots. The first is a screenshot of the coordinator, where the black points represent the positions of the boats, and where the coordinates indicate the current position of the cursor on the map. The second is a screenshot of the mobile application in "waiting mode", meaning that no move is required. In this case, the application shows a compass, the current position, the speed, the heading and the state of the application.





Lat: 46.26000
Lon: 6.18000
Vitesse (km/h): 20
Direction: 40° - NE
Etat: GPS DATA

Figure 5. Screenshots of the coordinator and the mobile application

6 GeoVTag framework

6.1 Overview

As we saw it in previous chapters, spatial messaging is not a new concept. But existing systems do not have a trust mechanism, thus preventing any serious application. And building a new trust engine for each application is certainly not the best solution, since the trust engine is the most complicated part.

Our solution to this problem consisted in building a framework that provides, among other things, a set of generic trust engines and a box providing geo-related tools. This framework, called GeoVTag, provides APIs in order to ease future development of applications using virtual tags.

The trust engines are generic and easily extendable. A designer of a new application will have to choose the trust engine whose behavior corresponds as much as possible to what he wants to do, extend it with rules in order to specify when a given tag should be considered as trusty (and therefore being returned to the requester) and how to update the trust values during a reviewing, and finally sets the values of the parameters in order to adapt the trust engine to a specific situation. He will therefore only have to code behaviors directly related to its application, leaving the framework doing all the job of maintaining and managing the trust information.

Our framework is divided in two parts, the server and the client. Remember that all the trust computations are made in the server, the client can actually be seen as a remote interface allowing to post new tags, to review them or to get a set of tags according too some criteria. We will still present the client part of the framework here since it provides a set of useful tools meant to facilitate further developments.

6.2 Server part

The server part of our framework is represented by figure 1. The Tools box is used by the trust engines and can also be accessed by the applications. It contains mostly geographical related tools, like methods allowing conversions between different geographical projections or methods handling tags of different formats.

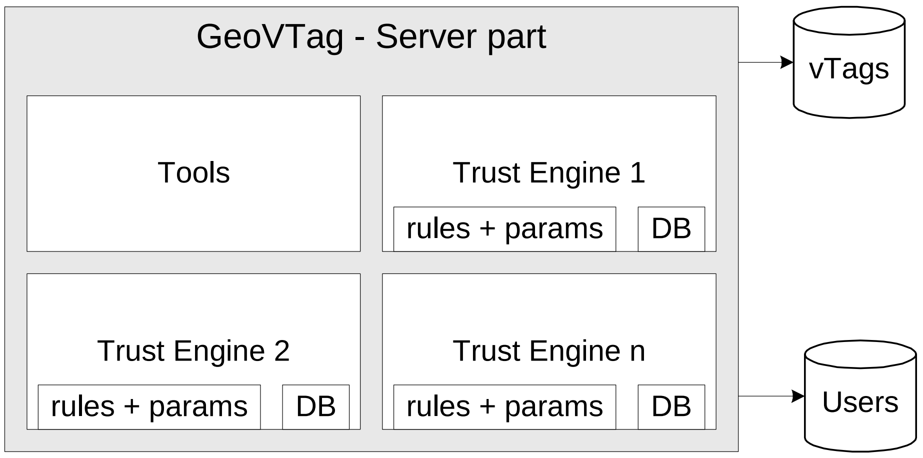


Figure 1. GeoVTag server part of the framework

All accesses to the two databases (vTags contains the virtual tags and Users contains the ID of the users as well as their trust relations) are done via the trust engines. Note that we use the word "database" in a very general way. It can of be any storage solution, including no permanent storage (information is kept in memory), flat files or a SQL standard database. Throughout this document, we will use the term "database" or its abbreviation "DB" to mention any storage system, and use the term "SQL database" or "SQL DB" if we talk about a "traditional" relational database using the SQL language to interact with.

Each trust engine is in a specific package and completely independent from the other ones. It is therefore easy to add new trust engines or simply new functionalities to this framework. In figure 1 we see that a trust engine contains some rules and parameters in order to adapt it to a particular situation, as well as a DB box. This DB box is the interface between the trust engine and the storage architecture. It only defines how the data must be handled and how extensions of this bloc will do the real job. For instance, in figure 2 we see three extensions, one that will save the data in flat files, one that will save the data in a SQL relational database, and one that will not save the data (all the data are maintained in memory).

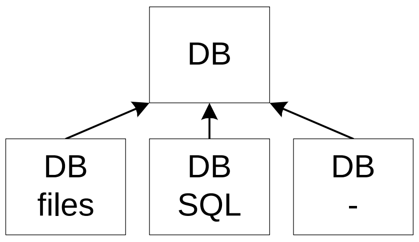


Figure 2. A DB box with its extensions

In our approach, we provide for each trust engine an abstract class that manages all the data but that does only define how exactly the data must be stored and retrieved from the storage architecture. By default, each trust engine comes with one implementation to store the data in flat files, and a second one that allows storing the data in a SQL database. These two classes can easily be adapted for a specific need (by creating a sub-class) and a completely different storage architecture can be set up. The abstract class gives all the requirements and describes what exactly the abstract methods are meant to do. A developer can then choose his storage architecture without having to study the code of the trust engine.

We chose flat files and SQL databases as default storage architecture. The reason is that in the flat files we can choose to save the data only from time to time, and therefore have very high performances. The SQL database is much slower, but guaranties that the latest information is saved and that the database will always stay in a consistent state. The user will therefore choose his storage architecture according to the security and performance requirements. He can also switch to one or the other easily, since it simply consists in giving another storage class to the constructor of the trust engine.

Currently, the framework contains 5 trust engines, called "BasicBin", "SimpleBin", "GenBin", "GenBin2" and "Ghosts". The first four are all binary ("Bin") ones, since it was, to our view, the most interesting kind of trust engines. The "Ghosts" trust engine deals with what we call ghost tags. Remember that a ghost is a tag that disappeared but that could reappear at the same place or in a close neighborhood. It describes an event or an object that is present only from time to time.

6.2.1 BasicBin trust engine

This trust engine allows any user to create or delete a vTag, which means that there is actually no trust computation. For simplicity and for easier understanding, we will still call "trust engine" all the architectures that allow managing virtual tags with a more or less complex algorithm to manage the trustworthiness of the information.

This trust engine doesn't contradict the general model where we asserted that it is not possible to delete directly a tag. Here "deleting" a tag consists simply in denying it, and this trust engine actually removes the tag as soon as someone denied it.

The tags handled by a BasicBin trust engine have the following fields:

- **id**: A unique ID for this tag. It is an unsigned integer > 0.
- **lat**: The latitude, a decimal number in [-90 .. 90], with a precision of 5 digits after the coma (to guaranty a precision of less than 1.1112 meters worldwide).
- **lon**: The longitude, a decimal number in [-180 .. 180], with a precision of 5 digits after the coma (to guaranty a precision of less than 1.1112 meters worldwide).
- **heading**: The heading where the tag is visible, usually the current direction of the author while he posted this tag. The heading is expressed in degrees, with North = 0°, East = 90°, South = 180° and West = 270°. This value is added to 1000 to mention that the tag has been recorded from someone coming in the opposite direction. For instance, a user that wants to tag an event that he sees on the road but that concerns drivers that travel in the opposite direction. The extra 1000

indicates therefore that the position of the tag is probably not very precise. The value -1 is used to indicate that there is no heading and that the tag must be visible from any direction. An heading is therefore an integer in [-1 .. 360] U [1000 .. 1360].

- **expiration**: The expiration time, expressed in milliseconds. After the expiration time, the tag is deleted. The value -1 is used to mention that the tag has no expiration time.

6.2.2 SimpleBin trust engine

The main idea of this trust engine is that we need one user to post a new vTag, but two to remove it. But if nobody confirms the tag, then only one user is needed to remove it. This simple trust engine protects the data against misusages. For instance, if a user sets by mistake a new tag, then only one user is needed to remove it. But if this tag is not a mistake and is confirmed by someone else, then two users are needed to remove it. Therefore if someone asks by mistake the removal of a tag, the tag actually does not disappear. The algorithm is very simple. Each tag contains a counter. When a user creates a new tag, its counter is set to 0. If someone confirms the tag, the counter is set to 1. If someone denies it (asking to remove it), the counter is decreased by one. And finally, a tag whose counter reaches -1 is deleted.

The tags handled by a SimpleBin trust engine have the following fields:

- **id, lat, lon, heading**: Same than for 6.2.1 - "BasicBin trust engine".
- **counter**: A signed integer that equals 0 at tag creation, 1 when someone confirmed the tag and -1 when one or two users denied it.

6.2.3 GenBin trust engine

A generic trust engine that creates trust links between the users and that excludes the malevolent ones. A history keeps all the ratings sorted in an inverse chronological order. Only the trusty vTags are returned during a request. GenBin considers that a tag is trusty if the two last friends (a user is a friend if his trust value is higher than 0) that rated the tag agreed with it.

If two successive users deny the tag, the trust engine gives a request-to-delete order to the tag. Technically it consists in filling the corresponding field with the current time. If a user votes positively for this tag, the request-to-delete is canceled. But if nobody votes positively, the tag is deleted by the trust engine after a certain delay.

Each time a user rates a tag, the trust engine updates the trust values of the author and the other users in the history. For instance, you will decrease the trust value of an author if you deny his tag, since you do not agree with it.

The tags handled by a GenBin trust engine have the following fields:

- **id, lat, lon, heading**: Same than for 6.2.1 - "BasicBin trust engine".

- **authorId**: The author of the tag. It can be the user's ID to mention that this given user is the author of the tag, or -1 to mention that the author is unknown, or -2 to mention that the author revoked its own tag.
- **speed**: The speed at which the user posted its tag. This value can be taken into account by the trust engine to compute the precision of the positioning. The value -1 is used if the speed is unknown.
- **creation**: The creation time, expressed in milliseconds. In GeoVTag times are expressed in the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 UTC.
- **expiration**: The expiration time, expressed in milliseconds. After the expiration time, the tag is deleted. The value -1 is used to mention that the tag has no expiration time.
- **reqDel**: This request-to-delete field contains the time a request-to-delete order has been launched, or -1 if no request-to-delete process is on.
- **history**: The history contains the user IDs of the reviewers and their marks, which equal "1" for the ones that agree with the tag and "0" for the ones that disagree.

A complete chapter, 7 - "Binary trust engine", describes this trust engine in more detail.

6.2.4 GenBin2 trust engine

GenBin2 is another generic trust engine, but that has some differences with the GenBin one. A first difference is that an application developer can specify the rules that decide whether a given tag is trusty or not. A second main difference is that the history contains only 2 places, which makes the trust engine more resistant against attacks (this will be discussed later in this document) but which also make it loose some information. The tags handled by a GenBin2 trust engine are the same as for the GenBin trust engine.

A complete chapter, 8 - "Towards a robust trust engine", describes this trust engine in more details.

6.2.5 Ghosts

As its name suggests it, this trust engine manages what we call "ghost tags" or simply "ghosts". A description on how a ghost works has been done in 4.3.1 - "Ghosts", and won't be repeated here. We will simply give here the fields of the ghosts that are treated by the Ghost trust engine:

- **id, lat, lon, heading**: Same than for 6.2.1 - "BasicBin trust engine".
- **authorId**: The author of the ghost. It can be the user's ID to mention that this given user is the author of the ghost, or -1 to mention that the author is unknown.
- **speed**: The speed at which the user posted its ghost. This value can be taken into account by the trust engine to compute the precision of the positioning. The value -1 is used if the speed is unknown.

- **creation**: The creation time, expressed in milliseconds. In GeoVTag times are expressed in the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 UTC.
- **activation**: The time this ghost has been activated.
- **delay**: The delay defines how long a ghost must stay active.

6.3 Client part

The client part of the GeoVTag framework consists of an API and some applications used to ease future developments. The API is developed in J2ME (Java Micro Edition) and provides geo-related tools, tag-related tools, as well as some basic functionalities that are missing in the micro edition of Java, like reading files, rounding numbers or internationalizing a class. A reference implementation allows testing the different functionalities provided by the API, a GPS simulator eases the debugging of new applications, and some test classes allow studying more carefully the behavior of some specific classes in particular situations.

The API is divided into three packages, called `geovtag`, `geovtag.gpsbt` and `geovtag.gpsint`. The two last ones allow a connection to a GPS and must be used in an exclusive way. The `geovtag.gpsbt` package allows a connection to an external Bluetooth GPS. If there is an internal GPS, it is the `geovtag.gpsint` package that will be used. The APIs provided by these two packages are almost identical, which means that the same application can run either on a mobile coupled to an external GPS, or on a mobile with an integrated GPS. This way of doing allows also integrating easily a future new package for phones that do not respect the java specifications and provide their own solution, like some Motorola phones.

We are not going to describe the behavior of each class from the `geovtag` package here, since they have been sufficiently documented directly in the source code and since the reference implementation can be used as an example to use them. We will rather concentrate here on one specific aspect: the computing of protected zones.

We mention also that the interested reader will find in 12 - "Annex 1: Technical details for the client side" some technical details, like the internationalization of the classes, the GPS simulator, or the test classes, and that a user's guide for the GeoVTag reference implementation can be found at 13 - "Annex 2: User's guide for GeoVTagRI".

6.3.1 Protected zones

In order to avoid making a connection every second to the server in order to check if there are new tags to display for the current position, the application downloads also all the neighboring tags and keeps them in memory. We call protected zone (PZ) the area that contains these tags. A new connection is then made only every given delay (typically 5 minutes) or if the user requests tags that are outside of the protected zone. This functionality is managed by the RemoteTagServer class. A protected zone is defined by 5 main attributes:

- **pzLat** and **pzLon**: Two numbers, the latitude and the longitude of the center of the PZ. A PZ is always a circle. A user can give its current position if he wants to see the tags around him, but can also give coordinates of another point to see the tags in another place.
- **radiusOut**: This is the radius of the external circle of the PZ. It defines the area where the tags have been downloaded.
- **radiusIn**: This is the radius of the internal circle of the PZ. This value is used as a trigger to ask a new connection to the server. Since connections to a remote server are not instantaneous, this internal circle allows the application to update its protected zone before the user exits the former PZ.
- **timeout**: This is the maximum delay between two connections to the server. Even if the user doesn't quit its PZ, a connection is made time to time in order to guaranty the freshness of the information.

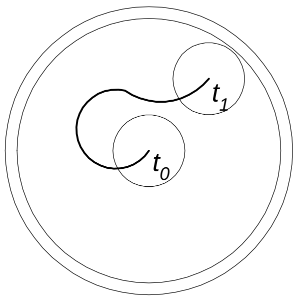


Figure 3. Moving in the same PZ

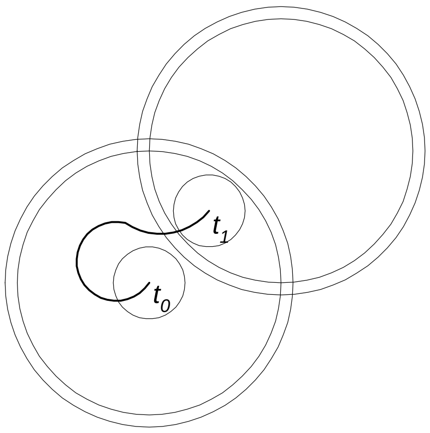


Figure 4. Computing a new PZ

The first time tags are requested, the algorithm will compute a PZ centered on the current position (or on the position where the tags are requested). But in order to limit the communications with the server, the next time a request is made, the PZ will be shifted according to the last moves of the user. In figure 3 we see an example. The user starts his application at time t_0. The little circle represents the radius of visibility (simply called radius), or the area where the tags are visible on the mobile device. The big inner circle is the internal circle of the PZ (called radiusIn), which acts as a trigger (when the user requests tags that are outside, a new PZ is computed). The big outer circle is the external circle of the PZ (called radiusOut), which is the area where the tags are actually present on the mobile device. While the user moves close to the center of the PZ, the tags are already in memory. If the user is still at the center of the PZ when the timeout is reached, a new connection is made to the server in order to update the tags, but the PZ stays at the same position (centered on the user). But if the user moved away, then the new PZ will be shifted in the same direction. For instance, if our user reaches the inner circle of the PZ at time t_1, a new PZ is requested and the latter will be shifted like it is shown in figure 4. The distance of the shift is proportional of the distance that the user moved since last update, so if he didn't move, the shift is null (and the new PZ is centered on the current position).

The RemoteTagServer class provides a isFresh method which return true only if the little circle is inside the outer circle of the PZ, if the timeout has not been reached, and if there is currently no other request to update.

7 Binary trust engine

We classified the trust engines according to the kind of tags they are going to deal with. We will now focalize on binary trust engines. We found this kind of trust engine particularly interesting since they have lots of practical applications despite the fact that there are simpler than the others. And, most of all, they allow us to fully exploit some particularities of spatial messaging systems when designing the trust engine, like the fact that it is difficult to increase ones trust value without bringing a real positive contribution to the system.

This chapter describes the GenBin trust engine that has already been introduced in a previous chapter. It starts by explaining more deeply the trust engine, then it explains how we are going to evaluate it, then it presents the results, and finally it shows how we used a widely deployed application in order to check that our simulator behaves in a way close to reality.

7.1 GenBin trust engine

7.1.1 Overview

The main idea of this generic binary trust engine is to remember only important or recent information, like it is done in human communities. vTags and users keep a history of their last or important transactions.

To know whether a tag must be shown to the user, the trust engine checks the *n* last reviews done by trustworthy users. A user is trustworthy if his combined trust value, computed as a mix of the trustor's opinion (based on former direct interactions) and the opinions of the trustor's friends (who ask their own friends, and so on until a certain level), is above a certain threshold. A trustor calls "friend" every user with who he has a good trust relationship, or better said, each user with a local trust value higher than 0.

When a user rates a tag, he updates the trust values of the author and the former reviewers according to rules and parameters that depend on the application. In certain cases, a review can be done on both directions. For instance an author can update the trust value of every reviewer that gives a positive rating, since they seem to share the same opinion about the tag.

7.1.2 A vTag in GenBin

We already presented the different fields that are present in a GenBin vTag in 6.2.3 - "GenBin trust engine". We are now going to speak more specifically about the history that each tag maintains.

The history is a two-column table containing pairs of user ID and corresponding vote. An example is given in figure 1.

ID	Vote
8	0
3	1
4	1
2	1

Figure 1. The history of a vTag

The lines are ordered in an inverse chronological order, meaning that the last user that voted for this tag is user 8. The vote can be either a "1", if the user confirms the tag, or a "0" if the user denies it. So we see that users 2, 4 and 3 agreed with the content of the tag, but later user 8 disagreed with it. The reasons can be either because user 8 is a malevolent user that wants to delete the tag, or, more probably, that the tag is outdated and needs therefore to be removed. If a user that is already in the history votes again for this tag, then his line is moved at the top of the table and the corresponding vote is updated.

When a user requests tags in a given area, the trust engine checks the vote of the two last friends (remember that a friend is someone in which we have a local trust value higher than 0) and if at least one of them voted "1", the tag is sent to the user. It means that even if someone denied the tag by mistake, the tag is still returned to people that are asking for it. This choice implies that we suppose that the price of a false positive (a tag that should not be sent is sent) if lower than the price of a false negative (a tag that should be sent is not sent), which seems to be the case in all the practical applications we thought about.

When the two last users denied the tag (they voted "0"), the tag gets a request-to-delete order. It means that the tag remains for the same amount of time than elapsed since its creation before being deleted by the trust engine. A tag that has been created a long time ago needs therefore more time to be deleted than a recent one. However, to avoid that an "old" tag needs too much time to be deleted we have a maximum delay. And, to avoid that malevolent users scan the network and deny the tags as soon as they appear, we added also a minimum delay. Since then, each new tag is at least present for a certain amount of time (the minimum delay), so even if malevolent users deny these tags, honest users will have time to confirm them (which will cancel the request-to-delete order) and by the same time decrease the trust value of the malevolent deniers.

7.1.3 A user in GenBin

A user is represented by an ID and a trust table. The trust table is a two-column table containing pairs of user ID and corresponding trust value. An example is given in figure 2. We see that this user has in his trust table two friends (users 3 and 7), one user he doesn't trust (user 8), and one user in who he has the same trust as for an unknown one (user 13).

ID	Trust
3	5
7	2
8	-3
13	0

Figure 2. The trust table of a user

After modifying the trust value of a user, the corresponding line is placed on top of the list, so that there are sorted in an inverse chronological order. Each trust value is simply an integer in the range [t_{min}, t_{max}] so that $t_{min} < 0 < t_{max}$. GenBin allows specifying rules to describe how a trust value must be changed according to a given situation. A typical case is to have a linear way to increase a value (for instance adding *n* when you agree with a tag) and an exponential way to decrease a value (for instance multiplying by *m* a negative trust value). And if $-t_{min}$ is much bigger than t_{max} (for instance t_{min} =-70 and t_{max} =5), then we imitate the human way of handling trust [47]: Trust takes time to be built, we forgive some small misbehaviors (exponential functions moves slowly at the beginning), but when we loose trust in someone (one big disappointment or lots of small disappointments) then it becomes very difficult to rebuild a good trust relationship. We avoid that malevolent users switch between good behaviors (in order to increase their trust value) and bad behaviors (in order to subvert the system).

It is important that our system forgives small mistakes in cases where the truth is unknown. Imagine that a user sees a tag, but the tagged object does not exist anymore. He will disagree with the author of the tag as well as with all the people that agreed. He will therefore decrease their trust values since they are perhaps spammers. But, most likely, the object simply disappeared in the meantime and they are not spammers. Our model is built to forget easily such mistakes, as long as they do not happen too often, but to decrease quickly the trust values of malevolent users.

The combined trust value of a user is relative and is computed by the following function:

$$combined\_trust = q * myOpinion + (1-q) * friendsOpinions \ , \quad q=[0..1]$$

It is a recursive function where *myOpinion* is the local trust value and *friendsOpinions* is the average opinion of the *n* first friends (where local trust > 0). These friends apply the same function, so they return a mix between their own opinion and the average opinion of their own friends. And so on until we reached the specified depth. This way of processing is fast (all the values are centralized) and gives a good idea of the global reputation of a user. Typically, if we choose *n*=10 (number of friends) and a depth level of 3, then we have already the opinion of $10^0 + 10^1 + 10^2 + 10^3 = 1111$ reliable people including ourselves, with more importance given to close friends. The higher is *q*, the more the user gives importance to his own value. In situations where people are susceptible of making mistakes, this value is usually quite small.

7.1.4 Trust updates

When a user votes for a tag, he puts his ID and his vote at the first line of the tag's history. This newly updated history is then analyzed by the trust engine, and the trust values of the users (that are in the history) are update according to their votes. For instance, if a user votes "1" and the two previous voters voted "0", the confirmer will decrease the trust value of the deniers. And perhaps increase the trust value of the author. The trust engine proposes a default behavior for each situation that can be adapted by the application developer in order to better meet the requirements of his application.

7.2 Validation process

We chose a speed camera tagging application to validate our trust engine. The first reason is because the topic is quite complex and interesting. Speed cameras can appear and disappear at any time, and it is not always possible to know if a false alarm is due to spammers or if it is actually the speed camera that just disappeared. The second reason is that it was very easy to find volunteers to test our system. We set up a simulator that allowed us to test different scenarios (spammers, users that try to delete all the tags...) as well as a widely deployed application used to confirm the results of the simulator.

7.2.1 Rules for the speed camera application

Each new user has an initial trust value equal to 0. A user is meant to send "1" if he sees a speed camera, or "0" if he gets an alarm but does not see any camera. If the application gets a "1" and there is no neighboring camera (less than 150 meters), it is considered as a creation of a new tag. If there is a neighboring camera, this "1" is considered as a confirmation vote for the existing one. A "0" is therefore considered as a denying of the tag. We extended our trust engine and wrote a set of rules. The syntax is the following: "Vote *the_vote* for *last_entries_in_the_history*". For instance, if we see "Vote 1 for 00", it means that the current user is confirming the tag (vote 1) and that the two last entries in the history where done by deniers (that voted 0).

- **Vote 1 for 1**. Confirming a tag. The historySize-2 first people that confirm a tag increase by 5 the author's trust value and the author does the same with these reviewers.
- **Vote 1 for 0**. The previous reviewer denied the tag, but it seems the speed camera still exists. Its trust value is decreased by 3. It is not reasonable to decrease by more than 3 since it can simply be a misuse (mixing up buttons...) of the application. And since there must be at least 2 successive reviewers that deny

the tag before a request-to-delete is made, this error will not harm the quality of the system.

- **Vote 1 for 00**. The two previous reviewers denied the tag, but it seems the camera still exists. This time the chance of being a misuse is reduced and this pattern could be considered as two malevolent users trying to remove the tag. Their trust values are updated like $t' = t * 1.5 - 5$, so that a misuse can be easily forgiven but if this behavior is repeated then the trust value falls quickly.
- **Vote 0 for 1**. The previous reviewer confirmed the existence of the speed camera but it seems that there is no camera anymore. It can reflect a normal situation (the camera simply disappeared), so the trust value should not be decreased too much. But it can also be the result of a spam attack. Since a spam attack is less dangerous than a deniers' one, we observed that decreasing the trust value by 1 in this case is not too penalizing for honest users, and still sufficient to exclude spammers in a reasonable delay. If the confirmer is not trusty (trust < 0), we also decrease the trust of the next confirmer, and so on until we cross a trusty voter (0 or 1). This avoids that a massive collusion of spammers needs too much time to be excluded.
- **Vote 0 for 0**. This case happens when a second user denies a tag. The two users increase mutually their trust value by 5.

These rules motivate the users' participation. Posting or confirming a tag increases trust relationships. We could think that it is not a good idea to deny a tag when the speed camera disappeared. It is true that in such a case we decrease (-1) the trust value of the previous reviewer who was probably an honest user. But on the other hand, we will build a bidirectional trust relationship with the second user that will deny the tag, and the increase of the trust values (2 times +5) compensates generously the former loss.

7.2.2 Parameters for the speed camera application

The GenBin trust engine can easily be configured through a set of parameters. We remind here that parameters only allow little modifications on how the trust engine behaves. A developer that wants to modify the trust computation policies will have to extend the trust engine and code the behaviors he wants to change. The parameters are the following:

- **historySize = 10**: It is the number of votes that a tag keeps in memory. A new vote will automatically erase the oldest one. If a user already voted for a tag, the old rating is deleted and the new one is put on top of the list. We chose 10 for this value, so we keep only recent information. This value could seem small, but is perfectly adapted to an environment where changes can happen very suddenly.
- **minDelay = 6 hours**. When a request-to-delete order is given to a tag, the latter stays usually for the same delay than the time elapsed since its creation. However, to avoid that malevolent users remove all new tags as soon as they appear, and thus escaping any trust decreases since the tag will be erased before

the vote of an honest user, this minimum delay indicates that a tag that get a request-to-delete order will stay at least for 6 hours before being erased.

- **maxDelay = 50 days**. To avoid that a long term tag takes too long to be removed, this value indicates that a tag that got a request-to-delete order will stay at maximum for 50 days before being removed.
- **trustTableSize = 1000**. The size for the trust tables. We estimated that 1000 should be enough since anyway a user that request trust information will also ask his friends, which are going to ask their own friends, and so on.
- **userOpinionWeight = 0.2**. While computing the combined trust value of one user, this is the ratio between the user's opinion weight and his friends' opinion weight. In this case, we sum 20% of our own opinion with 80% of our friends' opinion. Remember that our friends ask also their own friends, which means that each friend returns in fact 20% of his opinion combined with 80% of the opinion of his friends. We could think that 80% is too much for the friends, but we need to remember that we ask the opinion of maxFriends, so in fact if maxFriends = 10 and my opinion counts for 100% (as a reference value), then the opinion of one friend counts for 40% (0.8/0.2/10) and the opinion of a friend's friend counts for 3.2%. For comparison purposes, we note here that if userOpinionWeight = 0.15, then a friend's opinion counts for 56.6% and the opinion of a friend's friend counts for 4.8%, and finally if userOpinionWeight = 0.1 then a friend's opinion counts for 90% and the opinion of a friend's friend counts for 8.1%.
- **nbLevels = 2**. The number of friends-of-friends levels used when computing the combined trust value for one user. We chose 2 for the simulator for efficiency reasons, but in a real-life environment where the number of connections per second is more reasonable, we can easily increase this value. With 2 levels we combine our opinion with the one of our friends and the one of the friends of our friends.
- **minTrustValue = -70**. The minimum trust a user can have. This value must be bigger (in absolute) than the maxTrustValue if we want to be close to the human way of handling trust (it takes time to trust again a user that became untrusty), but not too big either otherwise a single user could sufficiently decrease the global trust value of a victim in order to exclude him from the system.
- **maxTrustValue = 5**. The maximum trust a user can have. This value must not be too big so that, like in the human model, even a trusty user can quickly loose his good reputation in case of bad behavior.
- **maxFriends = 10**. This value indicates to how many friends we ask advice when we compute the combined trust value for a given user.

7.3 The simulator

Our simulator randomly positions speed cameras on a road and simulates user’s cars navigating according to given scenario parameters. An additional user, whose behavior can also be completely specified, logs its observations and returns the number of true positives (alarm: yes, camera: yes), false positives (alarm: yes,

camera: no), true negatives (alarm: no, camera: no) and false negatives (alarm: no, camera: yes).
We model our road as a single way on a highway. Exits are numbered between 1 and n. Between two exits there is only one speed camera, numbered between 1 and n-1. So the camera c1 is between exits e1 and e2, the camera c2 is between exits e2 and e3, and so on. Figure 3 shows a road model.

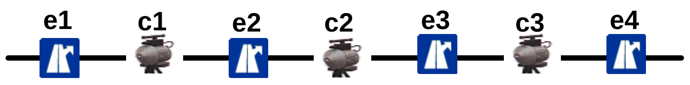


Figure 3. The road model

This model seems to be very simplistic. It is however sufficient to validate our trust metrics. Of course, we do not take into account some contextual information, like shadow areas (tunnels, urban canyons...) or what happens when the user posts a tag for the user driving in the opposite direction. These are more technical issues that need to be validated in the field and it is what we actually did with a real device in a real car. Since we can define the behavior of every user (where they enter and exit, how reliable they are by signaling speed cameras...) as well as the behavior of each speed camera (frequency of turning on, for how long...), we can precisely define which user drives in which area and how many speed cameras he is meant to cross on average. Our simulator accepts an input file that looks like this:

```
cam;1-4;8;15,10  // about three times a day, for 15 min, 10 min pause
cam;5-5;24;2,0   // about once a day, for 2 min, no pause
cam;5-5;240;3,30 // about once every 10 days, for 3 min, 30 min pause
usr;1-10;1-5;24;95;90  // once a day, 95% tp, 90% tn.
usr;1-1;3-5;240;80;75  // once every 10 days, 80% tp, 75% tn.
usr;11-15;1-10;1;10;10 // once every hour, 10% tp, 10% tn (hacker!).
usr;11-11;1-10;0;20;25 // once every minute, 20% tp, 25% tn (hacker!).
col;5-7;1-11;6;10;100  // four times a day, 10% tp, 100% tn.
spm;20-23;1-10;1       // once every hour
scn;100;2;run(24);pas(1,10);act(1,10,50,60)
scn;9;4;run(2400);pas(3,5);run(1);act(1,10,95,95);run(2);act(1,10,95,95)
```

- In the first line, "cam;1-4;8;15,10" means that cameras 1 to 4 have one chance out of 8 to become active within an hour, and when one becomes active then it stays active for 15 minutes. After it stays inactive (paused) for at least 10 minutes. Note that these cameras will on average become active less than 3 times a day, since they cannot switch to active while there are already active or paused. Precisely, these cameras will become active every 8+(15+10)/60 = 8.42 hours on average.
- The next two lines define two different behaviors for camera 5.
- In the fourth line, "usr;1-10;1-5;24;95;90" means that users 1 to 10 entry the highway at 1 and exits it at 5, that they run once a day and that they vote 95% of the time correctly when they signal the presence of a speed camera, and 90% of the time correctly when they cancel a camera.
- In the collusion line, "col;5-7;1-11;6;10;100", we deduce that users 5 to 7 are colluding by entering all at the same time on entry 1, exiting on exit 11, and

voting (all similarly) about all 6 hours with 10% of true positives and 100% of true negatives.

- In the spam line, "spm;20-23;1-10;1", we deduce that users 20 to 23 spam by entering all at the same time on entry 1, exiting on exit 10, and voting 1 about every hour at every speed camera place.
- The first scenario, "scn;100;2;run(24);pas(1,10);act(1,10,50,60)" contains 100 big loops and 2 small loops. The scenario itself will be executed twice, then the trust engine is initialized, and then we re-execute the scenario twice. And so on (100 times).
- run(t) means that the system will run for t hours (simulation time). Each minute, the go method of each camera and each user is called, allowing them to act according to their specified behaviors.
- pas(e1, e2) means that our test user will passively drive once from exit e1 to exit e2. Passively means that he does not vote. His observations are logged and printed.
- act(e1, e2, tp, tn) means that our test user will actively drive once from exit e1 to exit e2 and has tp (True Positive) chances (in %) to vote correctly if he sees a speed camera, and tn (True Negative) chances (in %) to vote correctly when he tries to cancel a speed camera that does not exist (anymore). His observations are logged and printed.
- Everything after a // is a comment and is ignored by the parser.

7.4 Results

We compare here our GenBin trust engine with two engines presented in 6 - "GeoVTag framework", BasicBin and SimpleBin. This permits to the reader to appreciate the efficiency of the GenBin trust engine. For the GenBin trust engine, we tested it once with fixed speed cameras (Gen_F), and once with mobile speed cameras (Gen_M). The only difference is that in Gen_M the tags are automatically removed after 6 hours.

Scenario 1

```
cam;1-10;0;9999999;0
usr;1-100;1-11;24;100;100
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,100,100)
```

	tp	fp	tn	fn
Basic	43030	0	0	56970
Simple	58720	0	0	41280
Gen_F	99811	0	0	189
Gen_M	96066	0	0	3934

Scenario 1 tests our trust engine when malevolent users try to remove all the tags. We have 10 speed cameras that are always turned on (they are fixed speed cameras), a hundred users that behave always correctly and five users that systematically try to cancel all speed cameras they cross. Each hacker runs on average 24 times more often than an honest user. In the results table we compare the BasicBin, SimpleBin and the GenBin trust engines. We used also the following abbreviations: "tp" means true positives (alarm: yes, camera: yes), "fp" means false positives (alarm: yes, camera:

no), "tn" means true negatives (alarm: no, camera: no) and "fn" means false negatives (alarm: no, camera: yes).

With the BasicBin trust engine, we see that there are more false negatives (alarm: no, camera: yes) than true positives (alarm: yes, camera: yes). This is normal since the malevolent users are driving more than the honest ones. But our GenBin trust engine eliminates quite well these malevolent users, since less than 0.2% (189 / 99811) of the speed cameras where not tagged when we mentioned them as fixed ones (Gen_F).

Scenario 2

```
cam;1-10;9999999;0;0
usr;1-100;1-11;24;100;100
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,100,100)
```

	tp	fp	tn	fn
Basic	0	20820	79180	0
Simple	0	35280	64720	0
Gen_F	0	268	99732	0
Gen_M	0	260	99740	0

Scenario 2 tests how the trust engine reacts against a spam attack. This time the cameras are always turned off and the malevolent users vote "1" for each speed camera position. Again, we observe a significant improvement with our new trust engine.

Scenario 3

```
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
scn;100;100;run(24);act(1,11,100,100)
```

	tp	fp	tn	fn
Basic	8705	143	90767	385
Simple	8635	769	90241	355
Gen_F	8775	705	90201	319
Gen_M	8682	245	90747	326

In scenario 3 we have 10 speed cameras that are turned on every 66 hours (48 + (360 + 720) / 60) for 6 hours, and 100 users that vote always correctly. We expected therefore with Gen_F similar results than for the SimpleBin trust engine, which seems to be the case. But if we tag the cameras as mobile ones (Gen_M), we observe an interesting improvement for the number of false positives.

Scenario 4

```
cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
scn;100;100;run(24);act(1,11,95,95)
```

	tp	fp	tn	fn
Basic	8423	294	90472	811
Simple	8626	807	90155	412
Gen_F	8948	857	89824	371
Gen_M	8702	303	90601	394

In scenario 4 the users are voting incorrectly 5% of the time. This figure is clearly overrated (according to the tests realized with FoxyTag where this number is less than 1% in practice), but it let us to prove that our trust engine is tolerant with unintentional incorrect votes made by honest users.

Scenario 5

```
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,100,100)
```

	tp	fp	tn	fn
Basic	3845	76	90801	5278
Simple	5041	121	90887	3951
Gen_F	8577	863	90080	480
Gen_M	8541	259	90776	424

In scenario 5 we added 5 deniers that try to remove all the tags they cross. The honest users are behaving correctly 100% of the time. We have clearly more false positives than for the BasicBin trust engine. This is normal since the deniers removed all the tags, whether there is a camera or not. If we compare the results with the ones from scenario 4 (for Gen_M), we see that our trust engine eliminates efficiently deniers.

Scenario 6

cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,95,95)

	tp	fp	tn	fn
Basic	3612	60	91000	5328
Simple	4970	136	90705	4189
Gen_F	8415	1340	89525	720
Gen_M	8353	254	90683	710

In scenario 6 the users vote incorrectly 5% of the time. Unfortunately, we observe for Gen_M that the number of false negatives increases (compared to scenario 5). It seems that 5% of incorrect votes is a critical limit for this scenario.

Scenario 7

cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,100,100)

	tp	fp	tn	fn
Basic	8781	17824	73124	271
Simple	8910	32855	58023	212
Gen_F	8687	1680	89217	416
Gen_M	7692	8655	82261	1392

In scenario 7 we replaced the deniers by a spammer team, who votes "1" at every speed camera position. The other users are voting correctly 100% of the time. We observe quite bad numbers for Gen_M. We first thought of a weakness in our trust engine, but further investigations concluded that it is actually the simulator that presents a weakness. The problem is that the positions of the cameras are always the same (which is not the case in reality), and that sometimes, by chance, a spammer really signal a new speed camera, which generously increases its trust value. In reality this would not be a problem, since signaling randomly a real speed camera at the right place is almost impossible.

We observe that if we use fixed speed cameras (Gen_F), we get better results. But this is related to the weakness of the simulator. The reason is simply that a tag on fixed speed camera lasts longer, so the spammers have less possibilities to tag correctly, by chance, real cameras.

Scenario 8

cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,95,95)

	tp	fp	tn	fn
Basic	8595	18699	72115	591
Simple	8793	34164	56835	208
Gen_F	8733	2212	88484	571
Gen_M	6983	9379	81663	1975

In scenario 8 the honest users are voting incorrectly 5% of the time. We face the same weakness as in scenario 7. We got therefore a bit worse results, since the honest users are less reliable.

7.5 Real life evaluation: FoxyTag

The simulator allows us to test the trust models, but how to be sure that our simulator acts in a way that is close to reality? To answer this question, we tested our model with FoxyTag [12], a collaborative system to signal speed cameras on mobile phones. The idea consists in posting virtual tags close to speed cameras in order to warn the other drivers. These users will then get an alarm when they are closer than 15 seconds to a critical point, and a red point locating the speed camera appears on their screen. You can signal a fixed speed camera by pressing the key "1" of your mobile phone, a mobile one by pressing key "2" and that a camera disappeared (you get an alarm but you do not see any speed camera) by pressing "0". You are also invited to signal speed cameras that have already been tagged; by confirming their presence, you create trust links with other users and get more reliable information. The system excludes automatically users that do not vote "like the others". Roughly speaking, the more you participate, the more the information you get is reliable. FoxyTag is a "collaborative" system. Tags posted by FoxyTag are directional. So, tags posted for users driving in the opposite direction won't be signalized to you. FoxyTag motivates neither speeding nor any other risky behavior, but allows the driver to concentrate on the road instead of having is eyes fixed on the speedometer, by fear of being flashed. We observe that drivers tend to brake suddenly when they see a camera (even if they are not speeding), which can provoke traffic jams or even accidents. FoxyTag signals in advance the presence of speed cameras, so that the driver has enough time to check its speed and adapt it if necessarily. A more technical description of this application can be found at [13].

FoxyTag allowed us to correct some bugs and inconstancies in our simulator, and allowed us also to better interpret the results. For instance, we discovered that real users mixed the buttons (the ones to confirm or remove the tags) less than 1% of the time. Since our trust engine has been tested with up to 5% of misbehaviors, we have the confirmation that our trust engine is tolerant enough with users making small mistakes.

FoxyTag allowed us to determine that the minimum number of meters to leave between two consecutive speed cameras is 150 meters. It means that if a user presses "1" on his mobile phone and there is already a speed camera with the same heading that is closer than 150 meters, then the sent "1" is considered as a confirmation for the existing tag (otherwise it is considered as a creation of a new tag). We started by calculating how the precision of the postings varies according to the speed. It is important to mention here that this precision varies because of two main reasons. The first is because a GPS is less precise when it moves. The second is that the application needs a given time in order to update the data provided by the GPS, so the faster the user moves, the bigger the distance between the real position and the one displayed in the application. Figure 4 shows a recording done by the GeoVTag Reference Implementation.

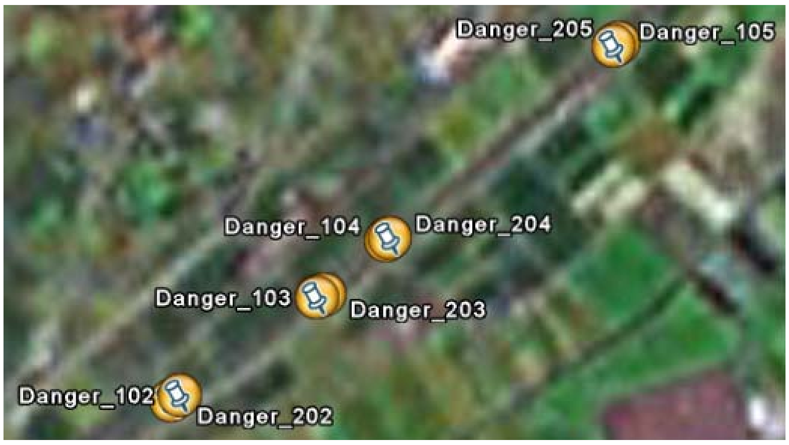


Figure 4. A dangerous road in Wallis (Switzerland)

We first recorded some dangers that are present on a given road (Danger_102, Danger_103 ...) without moving, and then we did the same recordings while moving at 80 km/h (Danger_202, Danger_203...). We repeated the experience several times and, according to the results, we concluded that 150 meters should be enough even on highways. We then set this distance parameter to 150 meters in FoxyTag and started to record the speed of several hundred users on a very loaded highway (in order to have records done with different speeds). We could thus confirm that 150 meters is sufficient, at least for the subset of users that have been taken for this study.

Another point that has been verified with FoxyTag is related to shadow areas. We call a shadow area a zone where the GPS is unable to get a position, like for instance in a tunnel. Our application keeps always in memory the last "good" position, so if a user tags a speed camera inside a tunnel the corresponding tag will simply be placed at the tunnel entry. The problem occurs in situations where a speed camera is close to a tunnel exit, like depicted in figure 5.

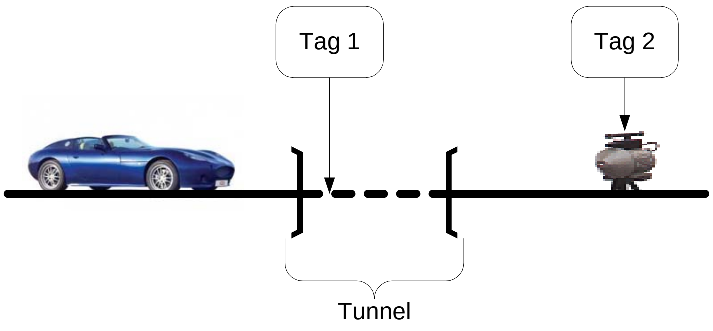


Figure 5. A speed camera close to a tunnel exit

At tunnel exit, a GPS device needs some time in order to get a position again. But this time depends on many factors, like the device itself (some device are faster than others), the weather (the signal degrades when it rains), the speed of the user, and some other reasons than we do not mention here. So even if all the users are honest and press their button when they cross the speed camera, the ones that won't get a fix (a good position from the GPS) before the speed camera will confirm the tag 1, and the ones that will get a fix before the speed camera will confirm the tag 2. This is currently an open issue for our trust engine. However, an analysis of the data collected by our users showed that this kind of situation is rare. And, more important, that the trust decrease provoked by these situations is not sufficient to penalize the users.

FoxyTag let us also discover that what the simulator classifies as "malevolent user to exclude" is probably "an honest user with a technical problem". We got for instance a few complains from users that have been excluded because considered as spammers by our trust engine. It came out however that they were all using a Sony-Ericsson W800 phone, and that they used the same button to change the current song as for posting a new tag... Another case was a user that confirmed the tags as soon as he saw the speed cameras, instead of confirming them when he crossed them. Since the distance between his confirmations and the speed cameras were often more than 150 meters, his confirmations where actually considered as new tag postings. We had therefore two tags per speed camera. What happened next is obvious: the other users denied his tags, which decreased his reputation, and after a while this user has been considered as a spammer by out trust engine and has been excluded.

After these anecdotes, we adapted our trust engine so that a "suspicious" user is simply put in quarantine (he can still use the system but he cannot tag anymore) and an alarm is sent to us. We then contact the suspects and only then decide if they deserve to be excluded or not. To date (2008-02-29), nobody has been excluded from the system and all the "suspects" could fully reintegrate the community after some clarifications with them.

7.6 Discussion

We set up a trust engine that deals with what we call the "uncertainty of the truth" or a situation where a trustor rates a trustee according to an observation that not necessarily reflects the truth. Our trust engine is generic and can be adapted through rules and parameters to any application using virtual tags. We chose the topic of speed camera tagging since it is a complex problem in terms of uncertainty (speed cameras can appear and disappear in a very unpredictable way) and since it was easy to find volunteers to test our application.

The results presented in this chapter where computed by our simulator, and some of them where compared with data collected by FoxyTag in order to make sure that our simulator behaves in a way close to reality. We observed that our trust engine excludes malevolent users but "forgives" small mistakes (due to the "uncertainty of the truth") and infrequent misuses (incorrect votes due a mix of the buttons) done by honest ones.

The main weakness we discovered in our work was directly related to the simulator. Since the positions of the speed cameras where always the same, spammers could by chance signal real cameras and then have their trust value generously increased. The second weakness was due to our trust engine and precisely with scenario 6. We saw that in case of a heavy attack, the honest users had to do less than 5% of incorrect ratings in order to keep the system reliable. In practice this is not really a problem since we observed that real people using the application do less than 1% of incorrect votes.

8 Towards a robust trust engine

We presented in the former chapter GenBin, a generic trust engine that resists against spam or denial attacks. We tested it through a set of 8 scenarios to prove its quality and we got good results, even under heavy attacks. However, our simulator supports only unstructured attacks, since the users are activated in a random manner. It is true that we can define precisely the behavior of each user, like saying where he drives, how often, and how he votes for speed cameras and with what accuracy, but then it is the simulator that randomly decides how the different drivers are activated. But an attacker that knows the trust algorithm as well as the rules and parameters can make what we call a structured attack. In a structured attack, the ratings are made in a very precise order in order to benefit from the weaknesses of the algorithm.

In this chapter we will present some weaknesses of GenBin and explore how we can avoid a structured attack. We will also describe a first version of an enhanced trust engine that looks simple and efficient, but that contains an inevitable weakness making it useless in a real situation. However, it will still allow us to define a clear syntax to express trust updates, and will lead us later to a new trust engine, GenBin2, which is (compared to GenBin) simpler, faster, and resistant against structured attacks. We will use the same simulator to compare GenBin2 with GenBin. Finally we will conclude this section by presenting some observations, some good and bad ideas we had during the conception of these trust engines, and discuss them in order to let the reader better understand the choices we made.

8.1 Weaknesses in GenBin

8.1.1 The 0-1 attack

This is a very simple structured spam attack where two users are colluding. The notation 0-1 means that the history contains a "0" in the first position and a "1" in the second position. To precise who posted them, we use the notation 0(U_2)-1(U_1), meaning that user U_1 voted "1" followed by user U_2 who voted "0". This history is represented by figure 1.

User	Vote
U_2	0
U_1	1

Figure 1. A history with the votes of two users

The attack consists simply in posting 0(U_2)-1(U_1) tags everywhere. A victim will get these tags since there is only one other user that denied it. And, according to the rules, he will even increase the trust of U_2 (since it is a false tag, the current user will deny it, which will increase the trust of the other denier). But, since U_1 is hidden by U_2 (according to the rules, the current user will only update U_2's trust), its reputation won't decrease, meaning that these two malevolent users can spam further one. Of course, the trust that U_2 has in U_1 will decrease, but since there is a minimum value, the global trust value (average of all the other users) won't be low enough to mark him as a cheater. And even if the trust of U_1 becomes too low (and then the tag is not returned anymore, thus ending the spam operation), the two spammers can simply switch their role. In this case it will be U_1 that increases his reputation thanks to the victims.

8.1.2 The 1_x-1_x-1_x-1_x-... attack

This structured denial attack consist in making existing true tags unavailable. It needs *S* colluding users, *S* being the size of the history. The first step consists in posting false tags everywhere in order to decrease the reputation of the malevolent users. These users become therefore untrustworthy. We use the notation 1_x to mention that this vote is done by a user that is untrustworthy in the eyes of the user that sees the tag (combined reputation). The attack consists then in confirming massively existing tags in order to fill the history with untrustworthy "1", like in 1_x-1_x-1_x-1_x-... Remember that the trust algorithm searches in the history the two first reliable friends, and if at least one of them voted "1", the tag is returned. But in this case there is no reliable friend and the tag is simply not returned. And since the victim does not modify the trust values in this case, the deniers are able to carry on their attack simply be voting again *S* times (in order to suppress the victim's vote) for this tag.

We could think that a solution to this problem is simply to return the tag if it is only filled by untrusty confirmers. But then the malevolent users will use the same technique to post false tags everywhere and launch a spam attack. Honest users will even decrease further more their trust values, which is an advantage in this case.

8.1.3 The 1_x-0-0 attack

The simplest way to harm the system that comes in mind consists simply in voting "0" two successive times (by two different users). A request-to-delete order is then given to the tag, meaning that the tag will disappear in a while. And, in the meantime, if the two deniers are trustworthy, the tag is not returned anymore to the requesters. People miss then existing tags. However this attack cannot be repeated, since a victim will vote "1" when he sees that a tag is missing, and this operation will strongly decrease the trust value of the two deniers. The attack fails, and if they carry on with other victims, their global reputation will fall under a certain limit and they will be excluded by the system.

There is however a solution to protect the trust values of the two deniers, and therefore be able to launch a structured denial attack. It consists in having a third colluding user that in a first time tries to decrease his own reputation (for instance by posting false tags everywhere), and in a second time uses his bad reputation to protect the deniers' trust values. The pattern of the attack is 1_x-0-0, to apply on an existing tag, like depicted in figure 2.

User	Vote
U_3	1_x
U_2	0
U_1	0
...	...

Figure 2. The 1_x-0-0 denial attack applied to an existing tag

Since the first user (U_3) of the list is untrusty, the victim asks the advice of the two next ones (U_2 and U_1). The tag is therefore not returned. And since in this pattern the trust values of the deniers are not decreased by the victim (see rules), the same attack can be replayed.

8.2 A "wrong good" simplified solution

We saw previously that GenBin is vulnerable under some structured attacks. And simply treating these specific cases is not sufficient, since the attacks described above are in fact a base for other ones. For instance, even if we deal with the 1_x-0-0 attack (for instance by decreasing the trust values of the two deniers), then an attacker team could use the pattern 0-0-1_x-0-0, which is an extension of the former one. Applied to an existing tag, this attack will prevent a victim from getting the tag. Of course, the victim will then decrease the trust value of the two first "0" in the pattern, but then we simply obtain the pattern 0_x-0_x-1_x-0-0. And then, since we search the two first trusty users, we end up to the two last "0" of the list and the tag is not returned. And these two "0" are protected, since a future confirmer will only decrease the trust value of the two first "0".

A first idea to solve this problem is to treat the pattern, and if the first users of the pattern are untrusty we carry on with the next users. For instance, in the 0_x-0_x-1_x-0-0 pattern, we decrease the value of the two first "0" and since they are untrusty we also decrease the value of the two next ones. This method is very efficient and allows to remove very quickly all the malevolent users. But it is so efficient that even honest users are excluded. The fact is that in a history filled by honest users it is normal to find different votes. First because when a tagged object disappears, the votes change too. Second because not everybody has exactly the same opinion at a given time about a given tag. For instance, a tag that is in a shadow area for one user will be seen differently than from a user that is not in a shadow area. And finally because sometimes even honest users make mistakes, for instance if they don't see a tagged object and therefore deny the tag attached to it.

A second idea consists in analyzing more precisely the content of the histories and detecting suspicious patterns. We did it through the use of regular expressions. We created an application called RegexSR [36] that allowed us to create in a WYSIWYG manner expressions and plugins in order to find easily all the possible patterns, but it came out that, first, their are too many possibilities, and second, that honest users are too much penalized. We got a confirmation from our simulator that excluded a good part of the honest users (in the simulator an honest user is one that votes correctly 95% of the time). We concluded that the bigger the history, the more structured attacks can be set up.

We decided therefore to reduce the size of the history. We decided to keep only the author and one or two successive deniers. We decided also to treat differently the author of a tag than the other voters, so the author is not added in the history anymore. The possible values in the history are therefore Ø (empty history), 0 and 0-0. If someone votes "1", the "0" are erased. If there are two "0", a request-to-delete order is given to the tag. If someone votes "0" and there are already two "0", the history remains unchanged. To know if a tag must be returned, we use the following rules:

History	**Rules**
Ø (empty)	if I trust the author, return true; return false;
0	if I trust the author, return true; return false;
0-0	if I trust booth deniers, return false; if I trust the author, return true; return false;

We execute the rules one by one until a condition make us to execute a "return true", in which case we return the tag, or a "return false", in which case we do not return the tag.

In these rules we see that if the history is Ø or 0, meaning that there is at maximum one denier, we simply return the tag according to the author's reputation. We do this since a "0" vote can simply be a mistake. However, if there are two deniers, it is either an attack or probably that the tagged object doesn't exist anymore. We then take into consideration the reputation of each user to decide whether the tag must be returned or not.

To avoid the 0(U_2)-1(U_1) spam attack described previously we set up two trust values, one for authoring tags and one for denying tags. We define the AT trust value (author trust) and the DT trust value (denier trust). In this case we decide that if someone votes "0" for this history, we increase the DT trust of U_2 (which is probably an honest user in most of the cases!) but we decrease the AT of U_1 (this is not a problem since an author usually increases his AT when other users confirm his tags). But if U_1 and U_2 are malevolent users, the decrease of U_1's AT trust will soon make this attack impossible since the tag is not returned if the author is untrusty. To show how we modify the trust values in each case, we define two functions. The first updates the AT trust value and is written like: UAT(U_1, U_2, *a*, *b*, *c*, *d*). It means that U_1 updates the local trust he has in U_2 as following: If the current trust of the trustee is equal or greater than 0, it multiplies the current trust by *a* and adds *b*, and if the trust of the trustee is negative, then it multiplies the current value by *c* and adds *d*. In a similar way, we define UDT(U_1, U_2, *a*, *b*, *c*, *d*) to update the DT trust. Finally we add also two functions, UAT(U_1, U_2, *a*, *b*, *c*, *d*, *C*) and UDT(U_1, U_2, *a*, *b*, *c*, *d*, *C*), where *C* is a specific condition that must be true in order to update the trust.

We show now how the trust evolves according to the votes. To mention the current user, the one who's voting, we write U_c. And to mention the author, we write U_a. In this case the condition *C* returns true only if there are at maximum *N* voters that already confirmed this tag.

History	**Rules if vote = 1(U_c)**	**Rules if vote = 0(U_c)**
Ø (empty)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)	UAT(U_c, U_a, 1, -1, 1.3, -1)

0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -2, 1.3, -2)
		UAT(U_c, U_a, 1, -1, 1.3, -1)
		UDT(U_c, U_1, 1, 5, 1, 5)
		UDT(U_1, U_c, 1, 5, 1, 5)

0(U_2)-0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -3, 1.5, -3)
	UDT(U_c, U_2, 1, -3, 1.5, -3)
		UAT(U_c, U_a, 1, -1, 1.3, -1)

This solution seems simple and reliable. It deserved to be presented here for several reasons. One of them is because it helped us to define syntactic rules to express how trust values must be updated, or how to obtain the trust value for a given user. A subset of these rules is presented throughout this section, and these rules will be used to set up and to present the GenBin2 trust engine. More precisely we will start from GenBin, but we will apply the syntax defined here to express the behavior of our new trust engine.

But, even if this solution seems reliable, it is actually not. Yes, it is more resistant than GenBin against some specific structured attacks, but it has also a major weakness. We called this weakness major, because one single user is able to subvert the system by a structured attack! The attack is done like this: The hacker first tags real objects, if possible in highly frequented places. Then he starts to post false tags everywhere. And that's all. What happens? People will confirm the correct tags, which will increase the hacker's AT trust. Since his trust is high, we will have victims that trust the false tags. But these victims will then decrease the hacker's AT trust. This means that they won't be spammed anymore, but the real true tags won't be returned either! So the spam attack is automatically transformed into a denial attack. And the victims of the denial attack will vote for the tag (they thing they are posting a new tag since they didn't got it, but in reality they are confirming the hacker's tag), which will again increase the hacker's AT trust. We then again switch to a spam attack. And so on. In conclusion, a single user can prepare an attack that will automatically switch between a spam and a denial attack according to his trust value. And since his reputation is automatically regulated, it will even be impossible to exclude him from the system!

8.3 GenBin2

8.3.1 From GenBin to GenBin2

After a few attempts like the ones described in the previous section, we ended-up with a new trust engine, called GenBin2, that is simpler, faster, and that resists structured attacks.

GenBin2 is based on GenBin. We find therefore the same way for computing a combined trust value (combining the local trust value with the one given by our friends), the same trust tables ordered in an inverse chronological order, and the same simulator to validate our models. GenBin2 can also be easily customized by defining the rules to apply when a trust value must be updated and by setting the values of the different parameters. There are however four main differences with GenBin:

- We use two trust values, the author trust value (AT), and the denier trust value (DT). To know if an author or a confirmer is reliable, we check his AT trust, and to know if a denier is reliable, we check his DT trust.
- We removed the author from the history. The author's opinion is therefore always treated separately from the confirmers or the deniers' one. We shorten also the size of the history, it equals 2.
- The trust update formula has been improved. It allows now to update differently a negative or a positive value.
- GenBin2 allows also to define the rules that decide whether a tag must be returned according to the content of the history.

8.3.2 Rules

The rules that define the trustworthiness of a tag for a given user, as well as the rules that define how the trust values must be updated, are written by the application developer. For our speed camera application, we wrote the following rules for a tag request:

History	Rules
Ø (empty)	if I trust the author, return true; return false;
1	if I trust the author, return true; if I trust the confirmer, return true; return false;
1-1	return true;
0	if I trust the author, return true; return false;
0-0	if I trust booth deniers, return false; if I trust the author, return true; return false;

1-0	if I trust the author, return true; if I trust the confirmer, return true; if I trust the denier, return false; return true;
0-1	if I trust the author, return true; if I trust the confirmer, return true; if I trust the denier, return false; return true;

These rules decide whether a given tag must be returned to the requester. We execute the rules one by one until a condition make us to execute a "return true", in which case we return the tag, or a "return false", in which case we do not return the tag.

We then defined also how the trust values must be updated. The next table shows the current history and shows how the trust tables of the author, the current user and the people in the history are updated according to the current vote ("1" or "0"). For instance, if the current user U_c votes 1 and the history is empty, then this user will increase the author's trust value if the condition *C* is met. In our case, *C* returns true only if there are at maximum *N* voters that already voted for this tag.

History	**Rules if vote = 1(U_c)**	**Rules if vote = 0(U_c)**
Ø (empty)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)	UAT(U_c, U_a, 1, -1, 1.3, -1)

1(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)	UAT(U_c, U_a, 1, -1, 1.3, -1)
		UAT(U_c, U_1, 1, -1, 1.3, -1)

1(U_2)-1(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)	UAT(U_c, U_a, 1, -1, 1.3, -1)
		UAT(U_c, U_1, 1, -1, 1.3, -1)
		UAT(U_c, U_2, 1, -1, 1.3, -1)

0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -1, 1.3, -1)
		UAT(U_c, U_a, 1, -1, 1.3, -1)
		UDT(U_c, U_1, 1, 5, 1, 5)
		UDT(U_1, U_c, 1, 5, 1, 5)

0(U_2)-0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -3, 2, -3)
	UDT(U_c, U_2, 1, -3, 2, -3)
		UAT(U_c, U_a, 1, -1, 1.3, -1)

1(U_2)-0(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_1, 1, -1, 1.3, -1)
		UAT(U_c, U_a, 1, -1, 1.3, -1)
		UAT(U_c, U_2, 1, -1, 1.3, -1)

0(U_2)-1(U_1)	UAT(U_c, U_a, 1, 5, 1, 5, *C*)
	UDT(U_c, U_2, 1, -1, 1.3, -1)
		UAT(U_c, U_a, 1, -1, 1.3, -1)
		UAT(U_c, U_1, 1, -1, 1.3, -1)
		UDT(U_c, U_2, 1, 5, 1, 5)
		UDT(U_2, U_c, 1, 5, 1, 5)

8.3.3 Additional rules

Rule 1: If you are in the first place of the history and you vote the same as previously, do nothing (no trust update and no modification of the history).

Without this rule a single user could delete a tag (by voting twice "0"). However, it is important to note here that this rule mentions explicitly that the two votes are the same. If you vote differently, the trust tables and the history are updated normally. We could thing that if someone votes differently, it was a mistake the first time and we can simply remove the former vote in the history and replace it by the new one. However, this behavior opens the door to a new structured attack: The hacker finds a tag whose history is $0(U_2)$-$1(U_1)$, and then simply votes alternatively "0" and "1". He first votes "0", so he increases his DT trust with U_2. Then he votes 1, which would erase his last vote, and then he votes again 0, which will again increase his DT trust with U_2. And so on. In short, this would allow anyone to get the maximum DT trust value.

Rule 2: If you are in the first place in the history and voted "0", then the tag is not returned.

This rule avoids that users are disturbed by an object that disappeared. For instance, if a user tagged an object, then you need two different users to give a request-to-delete order to this tag. But if you are the only one that votes for this tag, you will never be able to delete it, and the tag will always be returned to you.

Rule 3: If an author denies his tag, and if the history is either empty or contains a single "0", then the tag is removed immediately.

If you post a tag and nobody sees it, or if you post a tag by mistake and want to remove it, this rule avoids keeping a wrong useless tag. We see also that if the only person that voted for this tag denied it ("0"), then it is a good idea to remove the tag immediately. However we do not remove the tag if the history equals 0-0. The reason is because a malevolent user can set up a structured attack in order to increase his AT trust: He authors a new tag, wait for a while so that people confirming the tag increase his trust value, and then with the help of a friend denies the tag (0-0) and then revokes it. Since the tag disappears, he can post a new one at the same place and again benefit from the trust increases given by the *N* first users that will confirm the new tag.

8.3.4 Parameters

GenBin2 defines a set of parameters to personalize the behavior of our trust engine. We list them here, and give also the first values we chose for our speed camera application. These values have been chosen intuitively and are the ones that have been used in our simulations. However, we updated them later according to the results of the simulator and the observations done with our widely deployed application FoxyTag.

- **nbFirstConfirmers = 10**. It is the number *N* that is used in the condition *C* described in several previous sections. It indicates that only the 10 first users that confirm a tag increase the author's AT trust.
- **minDelay = 6 hours**. See 7.2.2 - "Parameters for the speed camera application".

- **maxDelay = 30 days**. See 7.2.2 - "Parameters for the speed camera application".
- **trustTableSize = 200**. The sizes for the AT and DT trust tables. See 7.2.2 - "Parameters for the speed camera application".
- **twl = -1**. This is an abbreviation of TrustWorthiness Limit. This novelty of GenBin2 avoids that the opinion of new users is not taken into account. The problem we observed with GenBin (applied to FoxyTag) is that new users usually start by voting for existing tags before posting their own ones. So as soon as someone didn't agree with them, their local trust value fell under "0". And since there where no other users that increased their reputation, their global reputation was also under "0" (even if actually very close to 0), which made them untrusty. But now, a user whose combined trust is above -1 is still considered as trusty.
- **userOpinionWeight = 0.2**. See 7.2.2 - "Parameters for the speed camera application".
- **nbLevels = 2**. See 7.2.2 - "Parameters for the speed camera application".
- **minTrustValue = -50**. The minimum AT or DT trust a user can have. See 7.2.2 - "Parameters for the speed camera application".
- **maxTrustValue = 5**. The maximum AT or DT trust a user can have. See 7.2.2 - "Parameters for the speed camera application".
- **maxFriends = 10**. See 7.2.2 - "Parameters for the speed camera application".

8.4 Results

To ease comparisons, we took here the same scenarios as the ones used to test the GenBin trust engine. We just added Gen2_F (fixed speed cameras) and Gen2_M (mobile speed cameras) in the tables. In this section we will only discuss the results of GenBin2, so please refer to 7.4 - "Results" for a description of the scenarios or for comments about the other trust engines.

Since the algorithms of GenBin2 are quite fast in comparison with GenBin, we could couple our trust engine to a quarantine list system. We decided that when the global AT or DT reputation (sum of the AT or DT local trust values given by all the other users) for one specific user falls under -20, then this specific user is put in quarantine and cannot vote anymore. We made this test after each small loop in the simulator. And since a small loop lasts 24 hours (simulated time) in our 8 scenarios, we can say that we tested the global reputation of all of our users once a day.

Scenario 1
cam;1-10;0;9999999;0
usr;1-100;1-11;24;100;100
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,100,100)

	tp	fp	tn	fn
Basic	43030	0	0	56970
Simple	58720	0	0	41280
Gen_F	99811	0	0	189
Gen_M	96066	0	0	3934
Gen2_F	99948	0	0	52
Gen2_M	92022	0	0	7978

In scenario 1 we observe an interesting improvement with our GenBin2 trust engine, since we have only 52 false negatives (instead of 189 with GenBin). This performance is mainly due to the quarantine list that eliminates malevolent users. However, if we tag these cameras as mobile ones (Gen2_M), we observe that the results are less good than for the Gen_M (well, this is done for comparisons purposes, since in practice these fixed cameras would be tagged as fixed ones). The explication is quite simple. In this scenario, all the users enter the highway at entry 1 and exit it at entry 11. So the first honest user that runs will tag all the speed cameras. But then the first malevolent denier that runs will decrease the trust value of this honest user, sufficiently to put him in the quarantine list. And since a tag stays only for 6 hours, this problem will be repeated several times, more precisely until the deniers are all excluded. But in the meantime, we will have about half of all the honest users in quarantine, meaning that our test user will double his number of false negatives.

Scenario 2
cam;1-10;9999999;0;0
usr;1-100;1-11;24;100;100
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,100,100)

	tp	fp	tn	fn
Basic	0	20820	79180	0
Simple	0	35280	64720	0
Gen_F	0	268	99732	0
Gen_M	0	260	99740	0
Gen2_F	0	925	99075	0
Gen2_M	0	840	99160	0

Scenario 2 shows that GenBin2 is less efficient than GenBin against a heavy spam attack. The reason is because we decided that if the history contains two "1" votes, we always return the tag, even if the author and the two confirmers are untrusty. Remember that we decided that in order to correct a weakness that we found in GenBin when facing a structured attack. But hey, if we compare to BasicBin or SimpleBin, we still are 20 or 35 times better!

Scenario 3
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
scn;100;100;run(24);act(1,11,100,100)

	tp	fp	tn	fn
Basic	8705	143	90767	385
Simple	8635	769	90241	355
Gen_F	8775	705	90201	319
Gen_M	8682	245	90747	326
Gen2_F	8759	748	90146	347
Gen2_M	8787	245	90619	349

Scenario 3 shows that when everybody is honest and votes correctly, GenBin and GenBin2 return similar results.

Scenario 4
cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
scn;100;100;run(24);act(1,11,95,95)

	tp	fp	tn	fn
Basic	8423	294	90472	811
Simple	8626	807	90155	412
Gen_F	8948	857	89824	371
Gen_M	8702	303	90601	394
Gen2_F	8806	802	89990	402
Gen2_M	8488	277	90856	379

Scenario 4 confirms that GenBin2 is tolerant with unintentional incorrect votes made by honest users.

Scenario 5
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,100,100)

	tp	fp	tn	fn
Basic	3845	76	90801	5278
Simple	5041	121	90887	3951
Gen_F	8577	863	90080	480
Gen_M	8541	259	90776	424
Gen2_F	8765	719	90102	414
Gen2_M	8761	262	90591	386

In scenario 5 we observe a little improvement for GenBin2. This is again due to the quarantine list, which eliminated all the malevolent users.

Scenario 6
cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
usr;101-105;1-11;1;0;100
scn;100;100;run(24);act(1,11,95,95)

	tp	fp	tn	fn
Basic	3612	60	91000	5328
Simple	4970	136	90705	4189
Gen_F	8415	1340	89525	720
Gen_M	8353	254	90683	710
Gen2_F	8637	795	90109	459
Gen2_M	8679	267	90604	450

Scenario 6, compared to scenario 5, shows that GenBin2 is more tolerant than GenBin when honest users that have to face a denial attack vote incorrectly 5% of the time.

Scenario 7
cam;1-10;48;360;720
usr;1-100;1-11;24;100;100
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,100,100)

	tp	fp	tn	fn
Basic	8781	17824	73124	271
Simple	8910	32855	58023	212
Gen_F	8687	1680	89217	416
Gen_M	7692	8655	82261	1392
Gen2_F	8073	3073	87754	1100
Gen2_M	8420	1345	89435	800

In scenario 7 we got better results with Gen2_M than with Gen_M, but worse results with Gen2_F than with Gen_F. The performance for the mobile speed cameras is simply due to the quarantine list that eliminates the spammers. For the fixed ones, the reason has already been mentioned in scenario 2; a tag that contains two "1" in its history is always returned, regardless of the reputation of the confirmers.

Scenario 8
cam;1-10;48;360;720
usr;1-100;1-11;24;95;95
spm;101-105;1-11;1
scn;100;100;run(24);act(1,11,95,95)

	tp	fp	tn	fn
Basic	8595	18699	72115	591
Simple	8793	34164	56835	208
Gen_F	8733	2212	88484	571
Gen_M	6983	9379	81663	1975
Gen2_F	7878	3471	87498	1153
Gen2_M	8085	1403	89695	817

In scenario 8, since the honest users vote incorrectly 5% of the time, we simply got a little bit worse results than for scenario 7. The comparison with GenBin done for the previous scenario can therefore also be applied here.

8.5 Discussion

In the previous section we always presented the results for fixed and mobile speed cameras in our scenarios. It is clear that this is done only to study our trust engines. For instance, in practice we wouldn't tag the fixed speed cameras of scenario 1 as mobile ones, so we wouldn't have 50% of our users in quarantine! And anyway, in practice we also re-compute the quarantine list after removing a malevolent user.

In general, we observe better results with GenBin2 than with GenBin. And, even more important, GenBin2 resists against structured attacks, which is not the case with GenBin. If we take also into account that the new trust engine (GenBin2) is simpler, easier to extend, and between 1.5 and 5 times faster (depends on the scenario), we can ask ourselves why should we keep this "old" (GenBin) trust engine? The answer is that GenBin still has a little advantage that is sufficient to keep it: it has a longer history, thus having more advices. And this can be useful in cases where the truth is not so obvious to guess than with speed cameras, where every honest user should either agree or disagree with the tag. We can have situations where even two honest users vote differently for the same tag. Since the history is longer, a third user has more chance to find in the history someone that votes in a similar way, for who he has a good trust value.

We will now finish this section by presenting some observations, some good and less good ideas we had during the conception of these trust engines, and discuss them in order to let the reader better understand the choices we had to do.

The first observation is that all the events need to be logged in the history, and that all the votes need to modify the trust values in a manner or another. We saw for instance while elaborating the trust engine presented in 8.2 - "A "wrong good" simplified solution" that if we do not log an event (like voting "1" when the history = 0-0), a malevolent user can always subvert the system. But if a user is logged and gives a wrong vote, then the following user will punish him by decreasing his trust value.

The second observation is that the author cannot simply be in the history and considered like another user. His role is really different, and we saw that we need to handle his trust updates separately.

The third observation concerns reverse trust updating. Remember that we call reverse trust updating the fact that a user updates the trust for someone that votes afterwards. For instance, in GenBin, the author increases the trust value of the *N* first voters that confirm the tag. But this is really dangerous. If a user is able to scan the network in an anonymous way, he can simply confirm all the tags that appear in order to see his reputation increased by the author. We tried to find some solutions against this, like posting randomly some false tags and excluding the users that will confirm them, but this is not that easy if we thing about practical cases. For instance, we can imagine that a competitor posts the same kind of tags, and our hacker can then simply check with the competitor system (if the hacker is not the competitor himself!)

whether the tag is a false one. For these reasons, we decide to improve the trust value only if the user really brings a contribution to the system. Of course, we can argue that the hacker can use the system of a competitor to post true tags and therefore increase his reputation. But in this case our hacker really brings a positive contribution that deserves a trust increase. Remember that in our system the trust values decrease much faster than they increase. And anyway, it is only his AT trust that will be increased, so he can use it to spam, but he cannot use it to deny tags, which is more harmful.

However, we kept one reverse trust updating in GenBin2. It is when two successive users deny a tag. In this case they mutually increase their DT trust value. But this time it is not risky, because it is impossible to know when a first user denied a tag since the latter is still returned to the requester. And denying randomly tags is far too risky for a hacker. Again, the only way to deny a tag and increase our own reputation is to make really the observation in the field, thus bringing a real positive contribution to the system.

The fourth observation is that we need two trust values, one for authoring or confirming tags, and one for denying them. Otherwise it is possible to do an easy structured spam attack, like described in 8.1.1 - "The 0-1 attack". And even if in this case we decrease the trust value of the author (it is the case in GenBin2 but not in GenBin), we saw that these two users can simply switch their role and carry on their spam. However, with two different trust values, this attack is not possible anymore. Even if they switch their role their AT trust values will decrease and the tag won't be returned anymore to the requesters. And if their AT trust values decrease too much, they will be excluded from the system, even if they have a very high DT trust value.

The fifth observation is that we need a condition *C* when updating the author's trust. We decided that only the *N* first confirmers will increase the author's trust. The reason is that an honest user that posted lots of tags could become malevolent. And if he has tags posted everywhere, other users will continue to increase his trust even if he uses his high reputation somewhere else to make spam. Remember also that a user do not know his own trust value. He does not know either how many people are voting for him, in other words he has no idea about his current value and no idea about how it evolves. It means that if his trust begins to fall quickly, he won't be aware of that, and if he continues to misuse the system he will be excluded from the system.

9 Conclusion

9.1 Contributions

This work brought to the scientific community mainly two contributions: a generic framework to handle virtual tags on mobile devices and a set of trust engines running on this framework. We saw that posting virtual tags was not a new concept, but that there where however no really serious or important application using it. To our mind, the reason was simply that it wasn't possible to trust this information. These applications could of course meet all the security requirements, like ensuring the authenticity and the availability of a message, or making sure that a single physical user owns only one or a limited number of different IDs, but this was not sufficient in an environment where people, that do not know each other, needed to evaluate the relevancy of the posted messages. We added therefore the missing piece of the puzzle, the trust, and took advantage of the recent technological improvements in the mobile phone and GPS world in order to test our models through widely deployed applications.

We built a generic platform designed to manage virtual tags called GeoVTag. In the eyes of an application developer, GeoVTag is seen as a set of APIs giving access to trust engines, geo-related tools and virtual tag storage architectures, as well as some applications designed to ease the testing of new applications, like a GPS simulator. GeoVTag is modeled in different blocs and can easily be improved, either by enhancing existing blocs, or simply by adding new ones.

We designed and modeled mainly two trust engines, GenBin and GenBin2. GenBin means Generic Binary trust engine. Generic because it can be configured through rules and parameters, and binary because it handles tags that have no content, or where the content is by default and known by the users, like for instance a speed camera warning application where each tag mentions a camera. These trust engines have been tested with a simulator and the results have been compared with the ones from FoxyTag, our real-life test application, in order to make sure that our simulator behaves in a way close to reality.

GenBin2 is an improvement of GenBin. It is faster and resists some structured attacks, but it maintains a smaller history of the latest events, since big histories offer more attack possibilities.

9.2 Future research directions

We saw previously that our work can easily be extended. We suggest two ways. The first consists in improving the framework itself. Our work was very application-oriented, and a deeper study of what a virtual tag exactly is could make our framework even more generic and new opportunities could appear. The second consists in adding new trust engines. We focalized on binary ones (handling tags that have no content) since it seemed to us that the majority of present and oncoming applications will use binary tags. However, we currently do not provide trust engines adapted for all the scenarios presented at the beginning of this document. For instance, in the scenario where people are meant to post virtual tags in restaurants in order to rate them and comment them, a trust engine should be able to use and interpret the content of the tags. Of course, designing such a trust engine is perhaps a PhD work in itself, but if someone does it, then it should be possible to integrate it in GeoVTag.

10 Glossary

- **AT**: Abbreviation of Author Trust. We often use this abbreviation as a qualifying word, so we will speak about "AT trust", even if the letter "T" in "AT" means already "Trust". If the term "trust" is ambiguous, we use sometimes "AT reputation", see reputation.
- **Author trust**: In systems where the users have different trust values, the author trust indicates how reliable this user is to post new vTags or to confirm existing ones.
- **Binary vTag**: A vTag that has not content, or where the content is predefined and unchangeable.
- **Binary trust engine**: A trust engine specialized in handling binary vTags.
- **Colored vTag**: A single-fact vTag with a limited number of different contents.
- **Colored trust engine**: A trust engine specialized in handling colored vTags.
- **Combined reputation**: The reputation that one user has in the eyes of another user and his friends. See reputation.
- **Combined trust value**: The combined trust value is the trust that one user has in another user, by combining its local trust value with the combined trust value given by his friends. It is a recursive function, meaning that a given friend will actually return a combination between his local trust value and the values returned by his own friends, and so on up to a certain level. The combined trust value is the one that is used in most cases in human communities, where people use their own experience as well as recommendations given by friends in order to determine how trusty a given target is.
- **Commenting a vTag**: Consists in adding a human-readable part to the vTag, either to complete the initial information, or to justify the corresponding rating, or booth.
- **Confirmer**: Someone who confirms a vTag.
- **Confirming a vTag**: Consists in giving a positive vote to a vTag, meaning that we agree with its content.
- **Denial attack**: Occurs when one or several malevolent users try to make unavailable existing vTags. This can be done simply by rating them negatively, but also by a structured attack.
- **Denier**: Someone who denies a vTag.
- **Denier trust**: In systems where the users have different trust values, the denier trust indicates how reliable this user is to deny vTags.
- **Denying a vTag**: Consists in giving a negative vote to a vTag, meaning that we do not agree with its content. In some case, it means that we want to erase this vTag.
- **DT**: Abbreviation of Denier Trust. We often use this abbreviation as a qualifying word, so we will speak about "DT trust", even if the letter "T" in "DT" means

already "Trust". If the term "trust" is ambiguous, we use sometimes "DT reputation", see reputation.

- **False vTag**: A vTag that has been posted by a malevolent user in order to harm the information provided by the system. A false vTag is considered as spam. It is the opposite of a true vTag.
- **Friend**: Someone in whom we have a local trust value higher than 0.
- **Global reputation**: The sum of the local trust values that all the other users have for the targeted user. See reputation.
- **Global trust value**: The sum of all the local trust values for a given user. It allows to find malevolent users and to exclude them. This must not be confused with the mean trust value.
- **Local reputation**: The reputation that one user has in the eyes of another single user. See reputation.
- **Local trust value**: Direct trust that one user has in another user, based only on former direct interactions between the two users.
- **Mean reputation**: The average of the local trust values that all the other users have for the targeted user. See reputation.
- **Mean trust value**: The average of all the local trust values for a given user. Despite the appearances, this value cannot be used to determine how reliable a user is. The reason is that this value changes if we add new users at the other side of the Earth, even if they do not interact with the existing community. To determine how reliable a user globally is, we use the global trust value.
- **Multi-fact trust engine**: A trust engine specialized in handling multi-fact vTags.
- **Mutli-fact vTag**: A vTag whose content is divided in several objective parts, so everybody should have the same opinion for each part.
- **Multi-opinion vTag**: A vTag whose content is divided in several subjective parts, so every user can give its own opinion about each part.
- **Multi-opinion trust engine**: A trust engine specialized in handling multi-opinion vTags.
- **Negative rating**: Same as negative vote.
- **Negative vote**: A vote that indicates that we do not agree with the content of the vTag.
- **POI**: Often used by GPS users, a Point Of Interest is a geo-related piece of information containing usually only 3 fields, the latitude, the longitude and the content as a string. A POI can be seen as a very simplified vTag.
- **Position-dependent trust engine**: A trust engine that takes into account how two or more vTags are related according to their position.
- **Positive rating**: Same as positive vote.
- **Positive vote**: A vote that indicates that we agree with the content of the vTag.
- **Rating a user**: Means updating his trust value.
- **Rating a vTag**: Same as voting for a vTag.
- **Reputation**: The reputation of a user is proportional of the trust that the other users have in him. If *A* increases the trust he has for *B*, then *B* increases his reputation in the eyes of *A*. The term "reputation" is mainly used when the term "trust" is ambiguous, like for instance in "the trust value of *A*" which can design the trustworthiness of *A* or the trust value that *A* has for another user.

- **Request-to-delete**: Giving a request-to-delete order to a vTag means that this tag must be deleted in a while if no operation cancels the order in the meantime. Typically, a trust engine gives a request-to-delete order to a vTag that gets some successive negative votes. Since the vTag does not disappears immediately, other users have a chance to give a positive vote, thus to cancel the request-to-delete order, and to decrease the trust value of the deniers.
- **Reverse trust updating**: A reverse trust update occurs when a trustee updates the trust value of its trustor. For instance, an author that rates its reviewers does a reverse trust update.
- **Reviewing a vTag**: Consists in rating a vTag, in commenting a vTag, or booth.
- **Shadow area**: A zone where a user cannot make out its current position. For instance, if the user's positioning device is a GPS, he will have shadow areas inside buildings since the GPS doesn't function well in covered places. A shadow area depends on different parameters, like the weather (GPS) or the quality of the positioning device, meaning that a shadow area can disappear for a given user and still be present for another one.
- **Single-fact vTag**: A vTag is one whose content is objective, so everybody should have the same opinion.
- **Single-fact trust engine**: A trust engine specialized in handling single-fact vTags.
- **Single-opinion vTag**: A vTag whose content is subjective, so every user can give its own opinion.
- **Single-opinion trust engine**: A trust engine specialized in handling single-opinion vTags.
- **Spam attack**: Occurs when one or several malevolent users add false vTags in the system, or when they prevent an outdated vTag to be deleted. This can be done simply by rating them positively, but also by a structured attack.
- **Structured attack**: An attack where the ratings are made in a very precise order in order to benefit from the weaknesses of the algorithm.
- **True vTag**: A vTag that has been posted by an honest user in order to enrich the information provided by the system. It is the opposite of a false vTag.
- **Trust engine**: A general term used to describe a system that manages vTags. It is called like this because the main part of a trust engine is usually a trust algorithm responsible to evaluate how relevant a given vTag is for a certain user. But a simple data engine without any trust algorithm will also be called like this, since it is arbitrary to decide whether a given trust algorithm is really one.
- **Trustee**: The person that is trusted. A trustor trusts a trustee.
- **Trustor**: The person who trusts. A trustor trusts a trustee.
- **Unstructured attack**: An attack done without taking care of the behavior of the trust engine. For instance rating negatively and randomly existing tags is an unstructured attack, or more precisely an unstructured denial attack in this case.
- **Voting for a vTag**: Giving to a vTag a mark that reflects how much we agree with the content of this vTag.
- **vTag**: Also called virtual tag, a vTag is a geo-related piece of information. More generic than a POI, a vTag can be extended in order to be adapted to any situation. Compared to a POI, a vTag can for instance cover an area, be visible

only at a certain altitude or for a certain heading, having an expiration time, and being reviewed by different users. It contains also information computed by a trust engine so that a given user can know how reliable the vTag is to him. The content of a vTag can be a simple string like in a POI, but can also contain binary data or be structured with XML or HTML for instance.

11 References

[1] Google Earth Community website, visited the 12th of October 2007: http://bbs.keyhole.com/

[2] French constabulary website, visited the 12th of October 2007: http://www.defense.gouv.fr/gendarmerie/

[3] Burrell, Jenna, Gay, Geri K. (2002): E-graffiti: evaluating real-world use of a context-aware system. In Interacting with Computers, 14 (4) p. 301-312

[4] Persson, P., Espinoza, F., Fagerberg, P., Sandin, A., and Cöster, R. GeoNotes: A Location-based Information System for Public Spaces, in Höök, Benyon, and Munro (eds.) Readings in Social Navigation of Information Space, Springer (2000)

[5] William G. Griswold, Patricia Shanahan, Steven W. Brown, Robert S. Boyer, Matt Ratto, R. Benjamin Shapiro, Tan Minh Truong: ActiveCampus: Experiments in Community-Oriented Ubiquitous Computing. IEEE Computer 37(10): 73-81 (2004)

[6] Book "Hacker's Guide, Edition DeLuxe", CampusPress, ISBN: 2-7440-1628-4

[7] John R. Douceur. The sybil attack. In Proc. of the IPTPS02 Workshop, Cambridge, MA (USA), March 2002.

[8] Michel Deriaz. What is Trust? My Own Point of View. In ASG tech report 2006

[9] P. Zimmerman. PGP User’s Guide, MIT, 1994.

[11] POIPlaces website, visited the 12th of October 2007: http://poiplace.oabsoftware.nl/

[12] Mappy website, visited the 12th of October 2007: http://www.mappy.com/

[13] M. Deriaz and J.-M. Seigneur, "Trust and Security in Spatial Messaging: FoxyTag, the Speed Camera Case Study", in Proceedings of the 3rd International Conference on Privacy, Security and Trust, ACM, 2006.

[14] N. Dimmock, "Using trust and risk for access control in Global Computing", PhD thesis, University of Cambridge, 2005.

[15] Secure website, visited the 12th of October 2007: http://secure.dsg.cs.tcd.ie/

[16] R. Guha, "Open Rating Systems", 1st Workshop on Friend of a Friend, Social Networking and the Semantic Web, 2004.

[17] N. Mezzetti, "A Socially Inspired Reputation Model", in Proceedings of EuroPKI, 2004.

[18] M. Deriaz, "What is Trust? My Own Point of View", ASG technical report, University of Geneva, 2006.

[19] S. Buchegger and J.-Y. Le Boudec, "A Robust Reputation System for P2P and Mobile Ad-hoc Networks", in Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems, 2004.

[20] D. Quercia, S. Hailes, and L. Capra, "B-trust: Bayesian Trust Framework for Pervasive Computing", in Proceedings of the 4th International Conference on Trust Management (iTrust), LNCS, Springer, 2006.

[21] Universal-locator website of , visited the 12th of October 2007: http://www.universal-locator.com/

[22] ViewMap website, visited the 12th of October 2007: http://www.view-map.com/

[23] Mobil-i website, visited the 12th of October 2007: http://www.michelderiaz.com/masters/duteil/

[24] Dejan and Hikari's website, visited the 12th of October 2007: http://www.michelderiaz.com/masters/munjinwatanabe/

[25] GeoVTag website, visited the 12th of October 2007: http://www.michelderiaz.com/Softs/GeoVTag/

[26] FoxyTag website, visited the 12th of October 2007: http://www.foxytag.com

[27] arx iT website, visited the 12th of October 2007: http://www.arxit.com/

[28] SNG website, visited the 12th of October 2007: http://www.nautique.org/

[29] Mobile tag website, visited the 12th of October 2007: http://www.mobiletag.com/

[30] MUSCLE project website, visited the 12th of October 2007: http://www.muscle-noe.org/content/view/39/64/

[31] John R. Douceur, "The Sybil attack", in Proceedings of the IPTPS02 Workshop, 2002

[32] Truesense website, visited the 12th of October 2007: http://www.truesenses.com/

[33] Michel Deriaz, "Trust and Security for Spatial Messaging", ASG technical report 06, University of Geneva, 2006.

[34] Mobilife website, visited the 12th of October 2007: http://www.ist-mobilife.org/

[35] Flickr website, visited the 12th of October 2007: http://www.flickr.com/

[36] RegexSR website, visited the 12th of October 2007: http://www.michelderiaz.com/apps/regexsr/

[37] Geoskating website, visited the 12th of October 2007: http://www.geoskating.com/

[38] Geotracing website, visited the 12th of October 2007: http://www.geotracing.com/

[39] P. Zimmerman, "PGP Users Guide", MIT, 1994. Available at http://www.pgpi.org/doc/guide/

[40] Wikipedia website, visited the 12th of October 2007: http://www.wikipedia.org

[41] National Marine Electronics Association website, visited the 9th of November 2007: http://www.nmea.org/

[42] Radar buster website, visited the 9th of November 2007: Website: http://www.radarbusters.com/

[43] Smart speed website, visited the 9th of November 2007: http://www.smartspeed.fr/

[44] Coyote website, visited the 9th of November 2007: http://www.moncoyote.com/

[45] Inforad website, visited the 9th of November 2007: http://www.gpsinforad.co.uk/

[46] GpsPasSion website, visited the 9th of November 2007: http://www.gpspassion.com/forumsen/topic.asp?TOPIC_ID=21763

[47] Book: "Trust Rules: How to Tell the Good Guys from the Bad Guys in Work and Life (Hardcover)", by Linda K. Stroh, Praeger Publishers

(August 30, 2007), 184 pages, ISBN: 978-0275998646

[48] John R. Douceur. The sybil attack. In Proc. of the IPTPS02 Workshop, Cambridge, MA (USA), March 2002.

[49] P. Zimmerman. PGP User's Guide, MIT, 1994.

[50] Blind signature algorithm, website visited the 9th of November 2007: http://www.schneier.com/book-applied-toc.html

[51] Fiat-Shamir protocol. Website visited the 9th of November 2007: http://www.cse.scu.edu/~tschwarz/coen350/zkp.html

[52] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigen-Trust

[53] Epinion.com. http://www.epinions.com/

[54] eBay website, visited the 9th of November 2007: http://www.ebay.com/

[55] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigen-Trust Algorithm for Reputation Management in P2P Networks. 2003.

[56] Prashant Dewan. Peer-to-Peer Reputations. Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04) IEEE.

[57] V. Cahill, et al. Using Trust for Secure Collaboration in Uncertain Environments. IEEE Pervasive Computing Magazine, July-September 2003.

[58] Michael Kinateder, Kurt Rothermel. Architecture and Algorithms for a Distributed Reputation System. 2003.

[59] Aameek Singh, Ling Liu. TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems. Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P'03). IEEE.

[60] Anwitaman Datta, Manfred Hauswirth, Karl Aberer. Beyond "web of trust": Enabling P2P E-commerce. Proceedings of the IEEE International Conference on E-Commerce (CEC'03).

[61] Stephen Weeks. Understanding Trust Management Systems.

[62] Mogoroad website, visited the 5th of December 2007: http://www.mogoroad.ch

[63] TomTom website, visited the 6th of February 2008: http://www.tomtom.com/

[64] Global Positioning System according to Wikipedia website, visited the 6th of February 2008: http://en.wikipedia.org/wiki/Global_Positioning_System

[65] Galileo according to Wikipedia, website, visited the 6th of February 2008: http://en.wikipedia.org/wiki/Galileo_positioning_system

[66] Tele Atlas website visited the 6th of February 2008: http://www.teleatlas.com/

[67] Google website, visited the 7th of February 2008: http://www.google.com

[68] Tiny tiny blog, visited the 7th of February 2008: http://www.alvafilm.ch/blog/tinytiny/?cat=3

[69] Pierpaolo Dondio and Stephen Barrett and Stefan Weber and Jean Marc Seigneur, Extracting Trust from Domain Analysis, a Study Case on the Wikipedia Project 3rd International Conference on Autonomic and Trusted Computing (ATC 2006) LNCS 4158, Wuhan, China, 2006, L.T. Yang et al., 4158, Lecture Notes in Computer Science, pp. 362--373, sep, Springer-Verlag

[70] Data mining according to Wikipedia website, visited the 11th of February 2008: http://en.wikipedia.org/wiki/Data_mining

[71] J.-M. Seigneur, "Trust, Security and Privacy in Global Computing", PhD Thesis, Trinity College Dublin, 2005.

[72] Google Earth website, visited the 11th of February 2008: http://earth.google.com/

[73] GPRS according to Wikipedia website, visited the 14th of April 2008: http://en.wikipedia.org/wiki/Gprs

[74] RFID according to Wikipedia website, visited the 14th of April 2008: http://en.wikipedia.org/wiki/Rfid

[75] WGS84 according to Wikipedia website, visited the 14th of April 2008: http://en.wikipedia.org/wiki/WGS84

[76] GSM according to Wikipedia website, visited the 14th of April 2008: http://en.wikipedia.org/wiki/Gsm

[77] ISO 8601 according to Wikipedia website, visited the 14th of April 2008: http://en.wikipedia.org/wiki/ISO_8601

[78] UTC according to Wikipedia website, visited the 14th of April 2008: http://en.wikipedia.org/wiki/Utc

[79] XTML according to Wikipedia website, visited the 14th of April 2008: http://en.wikipedia.org/wiki/Xhtml

12 Annex 1: Technical details for the client side

12.1 Internationalization

The GeoVTag API has been internationalized in order to be easily translated in other languages. All the classes that display information to the user, like a warning message or simply the label of a button, obtain their information in an external file that is the resources folder (called "res" in J2ME). For instance, the language file of the FileChooser class is the res/geovtag/FileChooser.rb file, since FileChooser belongs to the geovtag package. To show this class in French, a developer would create the res/geovtag/FileChooser_fr.rb file. The language code ("fr" in this example) is written according to the IS0 639-1 format. The RessouceBundle class is then responsible to show all the messages in the correct language. Note that if a language file is missing for a particular language, then it is the default file (the one without language code) that is used instead. The GeoVTag Reference Implementation can be used as an example to see how internationalization is achieved.

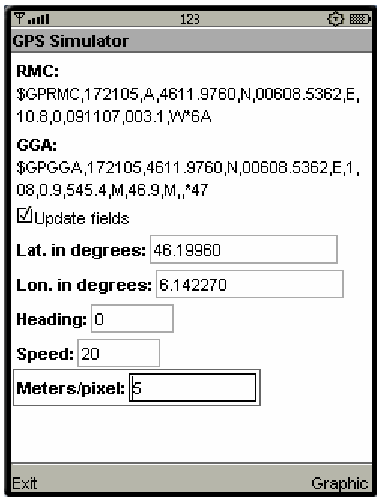
12.2 The GeoVTag Reference Implementation

In order to test the API and to show an example how to use it, the GeoVTag framework proposes a reference implementation. The main class is geovtagri.GeoVTagRI. Resources like some languages files, pictures and sounds are stored in the "res" directory.

This reference implementation works for device with an internal GPS as well as for devices paired with an external Bluetooth GPS. In order to switch between the two solutions, the developer simply has to change the package to import (`geovtag.gpsbt` or `geovtag.gpsint`).

12.3 The GPS simulator

The GPS simulator is a MIDlet designed to run on the emulator. It allows the user to enter a position and sends it via Bluetooth, in a similar way as any external Bluetooth GPS that we find in the market. Since the user can choose the values that are sent, this tool is very useful to test new applications.



RMC:

```
$GPRMC,172105,A,4611.9760,N,00608.5362,E,
10.8,0,091107,003.1,W*6A
```

GGA:

```
$GPGGA,172105,4611.9760,N,00608.5362,E,1,
08,0.9,545.4,M,46.9,M,,*47
```

☑ Update fields

Lat. in degrees: 46.19960

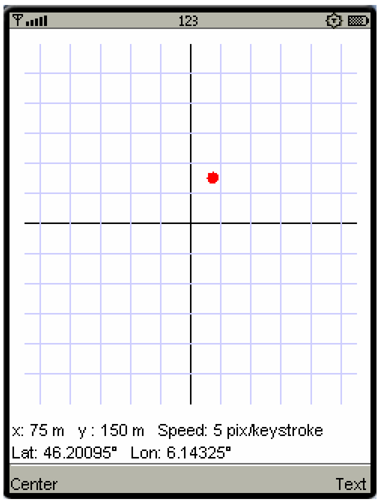
Lon. in degrees: 6.142270

Heading: 0

Speed: 20

Meters/pixel: 5

Figure 1. Textual screen



```
x: 75 m   y : 150 m   Speed: 5 pix/keystroke
Lat: 46.20095°   Lon: 6.14325°
```

Figure 2. Graphical screen

Figure 1 shows the textual screen of the GPS simulator. The user can enter the latitude, the longitude, the heading and the speed. The application creates then automatically two NMEA sentences (RMC and GGA) and sends them via Bluetooth. NMEA stands for National Marine Electronics Association (http://www.nmea.org/), and it defines a standard way to communicate data gathered from a positioning device. Messages are structured in different type of sentences, each giving a certain kind of information. So different positioning devices can understand different kind of sentence, and usually this can be configured via an external tool. We implemented in our simulator the two most frequently used sequences, RMC and GGA.

The last field in this screen, called "Meters/pixel" is used in the graphical screen (see figure 2) in order to choose what distance is represented by each pixel. The graphical screen allows the user to change the current position just by using the arrow keys. Two other keys, 1 and 3, allows to change the speed of the red point, or how many pixel this point moves each time the user presses a key.

12.4 Test classes

Inspired by the JUnit testing framework of the standard edition of Java, the `testclasses` package proposes some MIDlets used to test the other classes of the GeoVTag framework. For instance, the TestMenu MIDlet allows testing the Menu class. In case the Menu class is modified in the future, the test class helps the developer to check that the new modification behaves as expected and, even more important, that this upgrade does not harm the former functionalities.

13 Annex 2: User's guide for GeoVTagRI

13.1 General information

The GeoVTag Reference Implementation (GeoVTag RI) is an application that allows the user to publish anywhere on Earth virtual tags. Every user in the neighborhood of such a publication point will get the message. It is a kind of blog, in which editors and readers share the same physical place. GeoVTag RI allows tag edition and reading, but has also some additional functionalities like a digital compass, a tracker to record GPS data, a marker function that sends automatically time to time a tag, and a "goto" function that directs a user to a specific tag by showing graphically the direction to follow and the remaining distance. The GeoVTag RI is part of the GeoVTag framework and is given as an example of use of the GeoVTag API.

By default, tags are stored in a local file on the device. There is therefore no trust engine, and tag "posted" by the user are simply kept in memory and written in the local file if requested. The user can however select an URL in the parameters in order to store and retrieve tags from a remote server. In this case, a connection is made at least every five minutes in order to retrieve the latest tags around the current position and its neighborhood. This area is called Protected Zone (PZ) and a new connection to the server is made only if the user asks for tags that are outside this PZ, or if the last connection has been done more than five minutes ago.

At launch, the application shows a menu (see figure 1). Note that it is possible to get an error message at launch if the specified tag file (in the parameters) does not exist.

13.2 Tag function

The tag function shows graphically the positions of the neighboring tags (see figure 2). Each time a new tag enters in the covered area, it gets the smallest available number. This number determines its color, and allows the user to recognize it in the "details" screen. When a tag leaves the covered area, its number is made available for the next tag. The commands are:

- Details: Shows detailed information about the visible tags.
- Tag edition: Allows creating and sending a new tag.
- Save tags: Write all the tags in a file. This function is visible only if the application uses a local source of tags (tags stored in a local file).

- Update PZ: Update the tags that are in the protected zone (PZ). This function is visible only if the application uses a remote source of tags (tags stored on a remote server).
- Params: Allows the user to specify some parameters like the distance between two circles, whether the top of the screen represents the North Pole or the current heading, and whether numbers should be shown. If the alarm checkboxes are selected, then a sound (or/and a vibration) is played each time a new tag appears on the screen.
- Freeze GPS: Data that comes from the GPS device are not updated anymore.
- Unfreeze GPS: Cancels the "Freeze GPS" command.
- Test alarm: Tests the alarm.
- Back: Returns to the main menu.

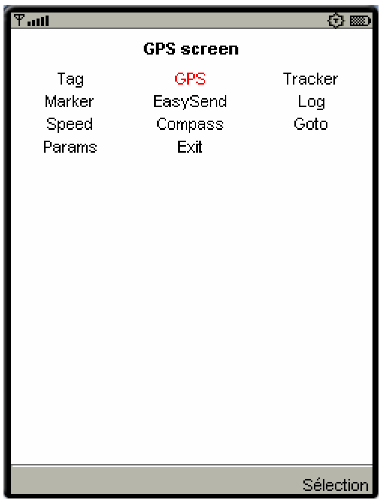
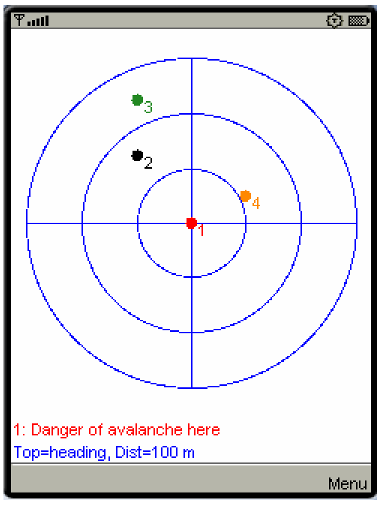


Figure 1. Main menu



1: Danger of avalanche here
Top=heading, Dist=100 m

Figure 2. Tags around

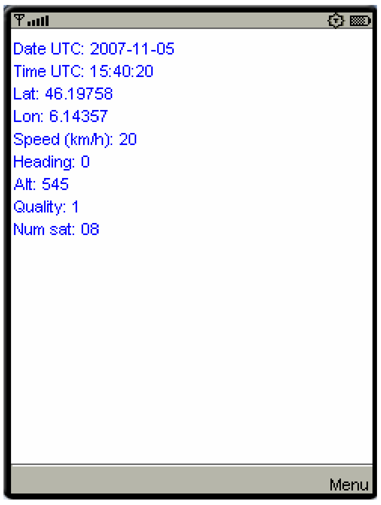
The details of the closest visible tag are printed at the last but one line, and the last line remind the user whether the top of the screen represents the current heading or the North Pole, as well as the distance between two circles. If the user does not move fast enough, the printed dots, whose positions are computed according to your current heading, become leaping. It is therefore recommended to choose the North Pole option.

13.3 GPS function

Displays information given by the GPS (See figure 3): The UTC date ("Date"), the UTC time ("Time"), the position ("Lat" and "Lon"), the speed ("Speed (km/h)"), the heading ("Heading", 0° = North, 90° = East, 180° = South and 270° = West), the altitude ("Alt"), the quality of reception ("Quality", 0 = no reception, 1 = normal

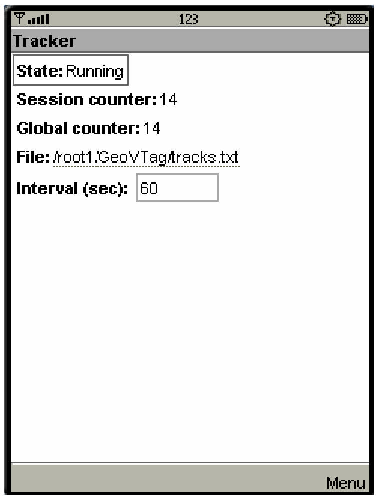
reception, 2 = differential mode), and the number of tracked satellites ("Num sat"). The menu provides two commands:
Connect: This command launches a search of all neighboring Bluetooth devices. Use the "Back" menu to cancel the operation, or the arrow keys to select a Bluetooth GPS from the returned list. The URL of the Bluetooth GPS is then saved so that the application tries to reconnect directly the next time it starts. This function is only available for the Bluetooth GPS version.

- Back: Returns to the main menu.



Date UTC: 2007-11-05
Time UTC: 15:40:20
Lat: 46.19758
Lon: 6.14357
Speed (km/h): 20
Heading: 0
Alt: 545
Quality: 1
Num sat: 08

Figure 3. Information from the GPS



Tracker

State: Running
Session counter: 14
Global counter: 14
File: /root1/GeoVTag/tracks.txt
Interval (sec): 60

Figure 4. Tracker function

13.4 Tracker function

Records in a file the GPS data every few seconds (see figure 4). "State" indicates the state of the process, can be "Not started yet", "Running" or "Stopped". "Session counter" counts the number of positions that have been recorded since the last time the data have been saved. "Global counter" counts the number of positions that have been recorded since the last time the application has started. "File" is the file that will contain the data. Click on it to browse the local file system. "Interval" specifies how many seconds the application pauses between two records. Note that this function uses the GPS as scheduler, which means that the real interval can vary with more or less one second (it depends when exactly the GPS gets an update of its data). The commands are:

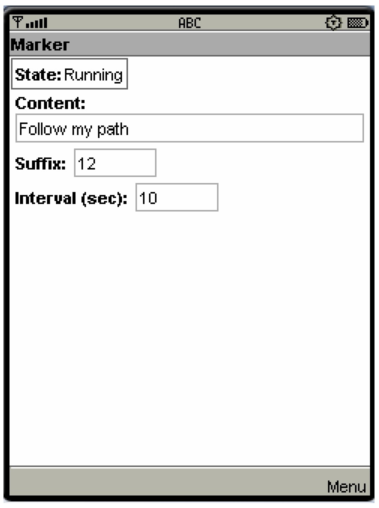
- Start: Starts the recording.
- Stop: Stops the recording.
- Record now: Records now the current data. Can be used while the process is running or not.
- Save: Saves the recorded data in the specified file, and clears the session counter.

- Main menu: Brings you back to the main menu. The process won't be stopped by this command.

13.5 Marker function

The function sends automatically a tag every few seconds (see figure 5). "State" indicates the state of the process, can be "Not started yet", "Running" or "Stopped". "Content" is the content of the tag. If "Suffix" is >= 0, then this number is added to each tag and automatically incremented. "Interval" specifies how many seconds the application pauses between two records. Note that this function uses the GPS as scheduler, which means that the real interval can vary with more or less one second (it depends when exactly the GPS gets an update of its data). The commands are:
Start: Starts the process.

- Stop: Stops the process.
- Send now: Sends a tag now. Can be used while the process is running or not.
- Main menu: Brings you back to the main menu. The process won't be stopped by this command.



Marker

State: Running

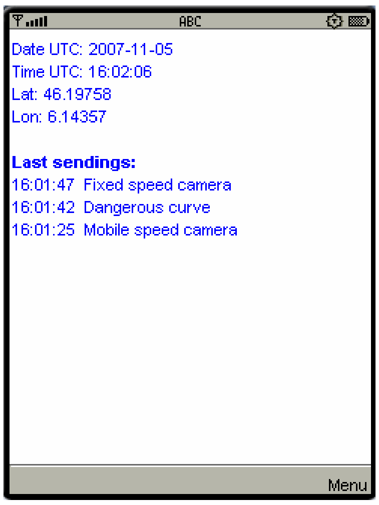
Content:

Follow my path

Suffix: 12

Interval (sec): 10

Figure 5. Marker function



Date UTC: 2007-11-05
Time UTC: 16:02:06
Lat: 46.19758
Lon: 6.14357

Last sendings:
16:01:47 Fixed speed camera
16:01:42 Dangerous curve
16:01:25 Mobile speed camera

Figure 6. EasySend function

13.6 EasySend function

Sends a tag each time the user presses a key (see figure 6). The user links each numerical key (0 - 9) to the text he wants to send each time he presses that key.

Remember however that if you use a local tag file then the tags are not automatically saved. The commands are:
Back: Returns to the main menu.
Captions: Allows the user to specify which text is linked to which key.

13.7 Log function

This function logs some events (see figure 7) in order to ease debugging. The commands are:

- Back: Returns to the main menu.
- Save: Saves the current content in the specified file.
- Clear: Erases all the current logged events.
- Browse: Allows the user to choose a file for recording the log events. This function is visible only if the filename has the focus.

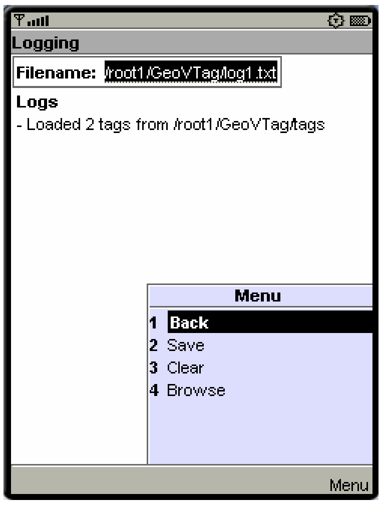

Logging
Filename: /root1/GeoVTag/log1.txt
Logs
- Loaded 2 tags from /root1/GeoVTag/tags

Figure 7. Log function

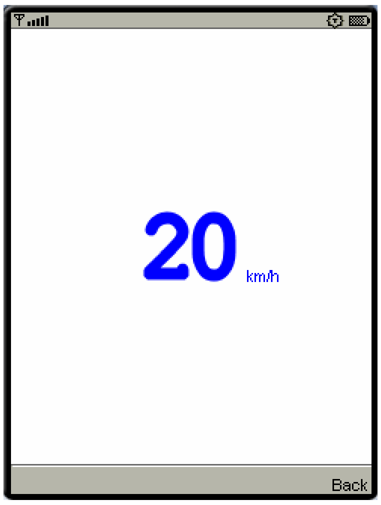


Figure 8. Speedometer

13.8 Speedometer function

This function indicates the current speed (see figure 8). The commands are:

- Back: Returns to the main menu.

13.9 Compass function

The red line heads for the North Pole (see figure 9). Note that it indicates the true North, and not the magnetic North, as traditional compasses do. The commands are:

- Back: Returns to the main menu.

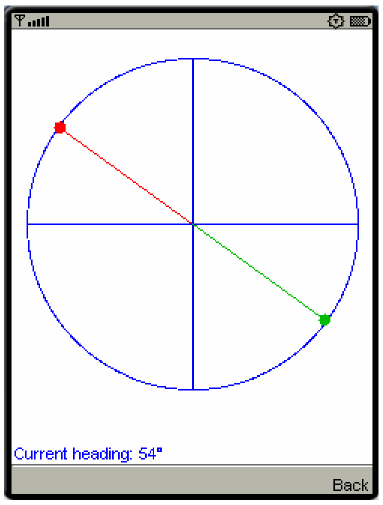


Figure 9. A compass

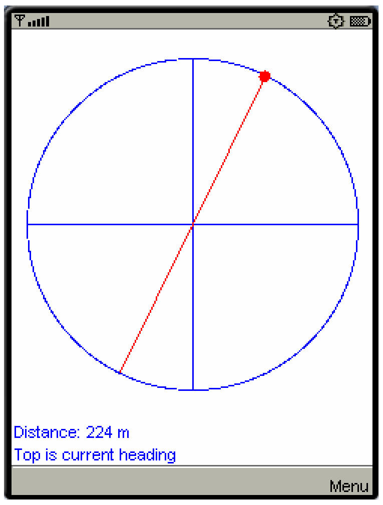


Figure 10. Goto function

13.10 Goto function

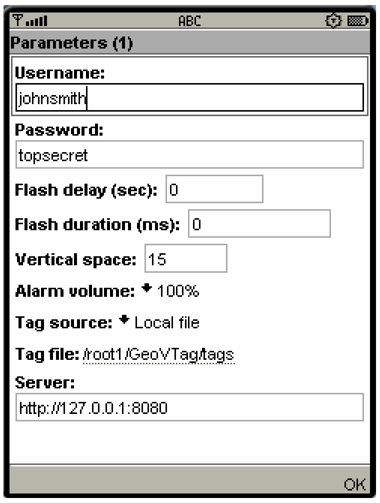
This function allows you to find a specific tag by giving its name or a part of its name (see figure 10). When you activate this function, you are prompted with a form asking you the name of the tag (or a part of its name) and the radius of search. The "Search" command looks then for all the available tags that contains the specified pattern (case insensitive) and which are not further than the specified distance. The result is presented in a list. Select one item. The program shows then graphically and continuously the direction to follow to reach the tag, and precisely at which distance it is. The first line of text indicates this distance between you and the tag, and the second line indicates whether the top of the screen is the North Pole or your current heading. The commands are:

- Details: Gives detailed information about the tracked tag.
- Params: Allows you to define whether the top of the screen represents the North Pole or your current heading.
- Back: Returns to the list resulted by the last search.
- Close: Closes this function and goes back to the main menu.

13.11 Parameters function

This function allows specifying the parameters for the application (see figure 11). The user can choose between 5 profiles. In each profile, the fields are:

- "Username" and "Password" can be used to be identified by the server.
- "Flash delay" and "Flash duration" avoid that your screen goes stand-by, which could block your application. Change them if your screen goes stand-by, blinks, or behaves strangely. For instance, most Sony-Ericsson phones work well with (10, 10), most Nokia phones work well with (1, 100), and most Motorola phones work well with (0, 0) or (2, 2100). However, with some Nokia S60 phones (N95, 6110 Navigator...) these parameters do not work (depends of the version of the firmware), use then (0, 0) as well as the S60SpotOn software. Note that with some phones you can deactivate the stand-by mode; use then (0, 0) for the parameters. Technically the application asks the screen to blink every "Flash delay" seconds during "Flash duration" milliseconds.
- "Vertical space" specifies the number of pixels between two printed lines.
- "Alarm volume" is the volume of the alarm, to combine with the volume setting of the device.
- "Tag source" specifies whether the tags must be posted and retrieved from a local file or from a remote server.
- "Tag file" specifies which tag file to use.
- "Server" specifies which server the application is connecting to.



Parameters (1)

Username: johnsmith

Password: topsecret

Flash delay (sec): 0

Flash duration (ms): 0

Vertical space: 15

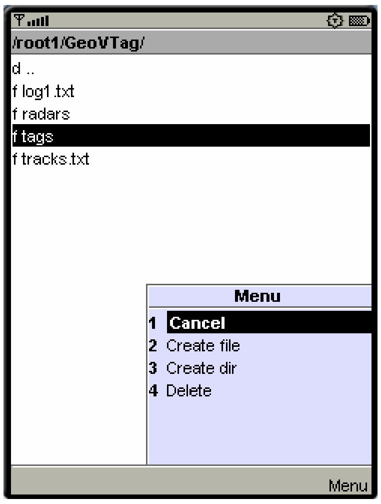
Alarm volume: 100%

Tag source: Local file

Tag file: /root1/GeoVTag/tags

Server: http://127.0.0.1:8080

Figure 11. Parameters



/root1/GeoVTag/

d ..
f log1.txt
f radars
f tags
f tracks.txt

Figure 12. Browsing the file system

13.12 Exit function

This function exits the application. It is important to remind here that local tags are not saved automatically when the application exits.

13.13 Browsing the local file system

The application includes a file browser that allows you to create new files or directories, delete them, and to select a file. It is used to select the local tag file in the parameters or to select where to save data from a track. The title of the screen indicates the current directory (see figure 12). "d .." means the parent directory. Actually, all the lines starting with "d" (and ending with "/") are directories, while line starting with "f" are files and lines starting with "r" are roots. Typically, a mobile phone has two roots, the first is the build-in memory of the phone and the second is an additional memory stick. We recommend saving tag files and track files in the memory stick rather than in the phone memory.

13.14 Saving parameters

The values entered in the Parameters screen, as well as the ones entered in different fields (mostly in parameters screens) are saved automatically in a MIDlet record store. Record stores stays even if the application is reinstalled. To delete the record store associated to GeoVTagRI, delete the application before reinstalling it.