TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair of Computer Networks

Thesis (Diplomarbeit), Single Volume Edition

TOWARDS A RELIABLE ARCHITECTURE FOR CROWDSOURCING IN CONTEXT OF THE MAPBIQUITOUS PROJECT

Dipl.-Inf. Tenshi C. Hara

Supervised by:
Dr.-Ing. Thomas Springer
Responsible docent:
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

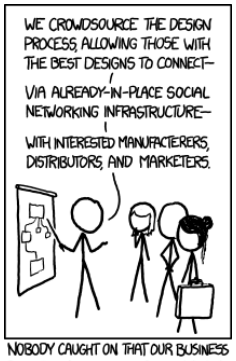
Submitted on 25 October 2012 (original deadline: 30 November 2012)
Date of the 'single volume edition' reprint: 28 January 2013

‘Isn’t it enough to see that a garden is beautiful without having to believe that there are fairies at the bottom of it too?’

Douglas Noël Adams

ABSTRACT

Extension of navigation applications to indoor use is currently under intensive research. There exist several promising concepts, for example the MapBiquitous project at TU Dresden. However, all concepts share the problem of ascertaining reliable map data and infrastructure information such as the position of WLAN access points. Further, maintenance and correction of data is challenging. A promising ansatz is to outsource ascertainment and correction of the data from the service provider to the service users. This ansatz is called '*crowdsourcing*'. There exist different derivates of crowdsourcing sharing diverse aspects, but not any commonly accepted definition. An attempt of a definition based on an existing, however very general definition is provided, resulting in a taxonomy classifying crowdsourcing by its user awareness and data immediacy. Utilizing the taxonomy conceived, an architecture extension compliant to German laws is introduced to MapBiquitous in order to support different classes of crowdsourcing. German law is keen on data protection. The extension conceived centres the 'indoor navigation server access network entity' (INSANE) which acts as a proxy for crowdsourcing communication between MapBiquitous clients and building servers, effectively concealing the crowdsourcing participants' identity from the crowdfunders. Addressing security and privacy issues, a user management and access control lists as well as concealing techniques are applied, and avoiding the obvious single point of failure at the INSANE, utilisation of a distributed hash table is proposed. The concept is exemplarily implemented and proven to be of good performance and scalability.



TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Systemarchitektur, Professur Rechnernetze

AUFGABENSTELLUNG FÜR DIE DIPLOMARBEIT

Thema: **Untersuchung von Methoden des impliziten Crowd-Sourcing für Location-based Services in MapBiquitous**

Name, Vorname	Hara, Tenshi	Studiengang	Diplom Informatik
Matrikel-Nr.	2936001	Projekt/Schwerpunkt	Mobile/MapBiquitous
Betreuer	Dr. Thomas Springer	Externer Betreuer	-
Beginn am	1. Juni 2012	Einzureichen am	30. November 2012

ZIELSTELLUNG

MapBiquitous ist eine Plattform zur Bereitstellung integrierter ortsbezogener Dienste für den Innen- und Außenbereich. Das Konzept beruht auf einer verteilten Infrastruktur von Gebäudeservern, die geometrische Daten der Gebäudestruktur, semantische Gebäudeinformationen, Daten zur Positionsbestimmung im Gebäude und Routinginformationen bereitstellen. Es bedarf eines hohen Aufwandes, um diese Daten zu erfassen und kontinuierliche Änderungen zeitnah zu integrieren. Die gilt etwa für Fingerprints, Wegpunkte und Pfade zur Navigation, die geometrische und semantische Beschreibung von Gebäudestrukturen sowie Points of Interest im Innen- und Außenbereich. Ansätze des Crowd-Sourcing, also der Einbeziehung einer breiten Nutzerbasis, könnten den Aufwand für die Datenerfassung und –aktualisierung wesentlich reduzieren und damit die Verfügbarkeit aktueller Gebäudedaten wesentlich erhöhen. Neben der Erfassung von Daten durch Beobachtung des Benutzers spielen auch die Validierung dieser Daten sowie Aspekte von Sicherheit und Privacy eine wichtige Rolle.

Ziel der Diplomarbeit ist die Untersuchung von Methoden des Crowd-Sourcing zur Erfassung und Aktualisierung von Gebäudedaten. Der Schwerpunkt der Betrachtungen liegt dabei auf automatischen Verfahren. Nach einer breiten Betrachtung möglicher Ansätze soll sich die Konzeption auf die Erfassung einer ausgewählten Kategorie von Daten konzentrieren. Die entwickelten Konzepte sollen auf der Basis von MapBiquitous prototypisch umgesetzt und unter Einbeziehung von Nutzern evaluiert werden.

SCHWERPUNKTE

- Recherche verwandter Arbeiten zum Crowd-Sourcing für Location-based Services
- Konzeption von Methoden zum impliziten Crowd-Sourcing für ausgewählte Daten
- Prototypische Umsetzung der Konzepte
- Erarbeitung einer Evaluationsmethodik
- Evaluation und Bewertung der Ergebnisse

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill (betreuender Hochschullehrer)

CONFIRMATION OF AUTOGRAPHIC WORK

Accordant §11, paragraph 6, sentence 2 of the 'Prüfungsordnung für den Studiengang Informatik an der Technischen Universität Dresden in der Fassung vom 11. Oktober 2004', I confirm that I independently prepared the following parts of this thesis:

- the introduction and motivation,
- the preliminaries,
- the summary of existing crowdsourcing definitions (not to be mistaken with the existing concepts, solutions or ansatzes) and attempt at a compatible definition with a corresponding taxonomy of crowdsourcing,
- the requirements review, especially the legal aspects of data protection issues as well as the data actually required by a crowdsourced MapBiquitous,
- within the crowdsourced optimization concept: MapBiquitous' server-side of the crowdsourcing architecture, including but not limited to the INSANE as well as the crowdsourced building server,
- within the proof of concept: the server-side implementation of the crowd-sourced optimization concept as well as the surrogate for the directory server, and last but not least
- any required adjustments to the corporate design TEX-template of the TU Dresden.

The other parts were prepared exclusively by and/or as a group-task in close liaison with Gerd Bombach, who prepared his assignment paper (Belegarbeit) with matching topic (explicit crowdsourcing; MapBiquitous' client-side) in parallel to this thesis.

Further, I confirm that I only used the references and auxiliary means indicated within the footnotes and at the end of this thesis.

Dresden, 25 October 2012

CONTENTS

DEFINITIONS AND THEOREMS

FIGURES

TABLES

INTRODUCTION AND MOTIVATION

'Biology and computer science – life and computation – are related. I am confident that at their interface great discoveries await those who seek them.'

Leonard Adleman

Ubiquitously one can find location-based services. One does not think about the actual location-based service, rather one uses it without consideration. For this, cellular phones have evolved from only providing mobile access to telephony services to providing a wide range of additional features: The quick and swift navigation to a point of interest or a price comparison while looking at an article in a shop. Nowadays, the mobile phone is frequently used for such tasks; hence, the term 'smartphone' has emerged. But, in order to use these services, not only must the service be usable and accepted by users; beyond that, a well founded base of data is required. The obvious question at hand is, where does the data come from? Should it be exclusively provided by the owner of a building or shop, bearing the risk of presenting outdated data and thus leading to customers' swizz? Or should it rather be provided by a third party, allowing the owner to focus on the primary business? Maybe a mixture would be good, allowing the owner to review results presented by the third party? And in general, how is the data maintained; how to keep it up to date? – A possible solution might be to involve the user of a location-based service into updating and correcting the data. This concept is well known and established as 'crowdsourcing' and shall be of focus within this thesis (Diplomarbeit). The user of the location-based service consciously or unconsciously, continuously or intermittently provides data to the location-based service. The data provided is either current, new or updated.

There exists the *MapBiquitous* project at the Technical University of Dresden, a project designed to provide feasible indoor navigation services, exemplary implemented on Android-driven smartphones. Obtaining and maintaining data, such as information on points of interest or WLAN fingerprints, has proven to be difficult in context of the existing MapBiquitous architecture; therefore, it is desirable to amend the existing architecture with crowdsourcing capabilities, either proving or disproving feasibility of crowdsourcing in the context of indoor navigation services.

As there is no common consensus on what crowdsourcing actually is and how it can be classified, a highly motivated desire to define a generally usable taxonomy of crowdsourcing exists.

The remainder of this thesis will continue in four parts; at first a propaedeutics part containing four chapters, second a proceedings part with another four chapters starting on page 64, thirdly an appendix containing important appendices starting on page 144 and lastly further appendices in the supplementary. In the propaedeutics part, the first chapters will provide some mathematic and general fundamentals which the author has identified as required preliminary knowledge. It will conclude with an overview of the current state of the MapBiquitous architecture in section 1.3. The fundamentals are followed by an attempt of providing a commonly acceptable definition of crowdsourcing and a taxonomy of the same. Chapter 2 will also analyse crowdsourcing in the context of social sensing and provide some information on terms such as 'explicit crowdsourcing' as well as 'implicit crowdsourcing'. In the then following chapter 2, two existing examples of crowdsourcing are presented, concluding the propaedeutics part. With respect to amending the MapBiquitous architecture, chapter 5 as proem of the proceedings part will review the requirements to such amendments, with a special focus on data protection issues in Germany in the first section of the chapter. Concluding the requirements review, a crowdsourced optimisation concept to amend the MapBiquitous architecture is presented and discussed in chapter 6. The first section of the chapter summarises Gerd Bombach's work [Bom12] on the client's side, while section 6.2 presents the concept on servers' side.

Comment

> *This thesis (Diplomarbeit) and Gerd Bombach's assignment paper (Belegarbeit) [Bom12] are being prepared in parallel, working on the same topic as a team effort. While this thesis focuses on implicit crowdsourcing aspects, which can mainly be dealt with on servers' side, [Bom12] focuses on explicit crowdsourcing, which – as Definition 2.2.1 will show – obviously includes user awareness, so it should be dealt with on the client's side. Therefore, it has been decided to divide the workload into client side and server side aspects. Accordant §11, paragraph 6, sentence 2 of the 'Prüfungsordnung für den Studiengang Informatik an der Technischen Universität Dresden in der Fassung vom 11. Oktober 2004', the individual contributions are clearly emphasised in the confirmation at the beginning of this thesis.*

After presenting both sides of the proposed optimisation architecture, the interplay of the two is discussed. The proof-of-concept implementation is presented and discussed in the following chapter 7, where the focus is set on difficulties when implementing the conceptual design into actual executable code, be it in Java for the Android ADK or Apache Tomcat, or in PHP for Apache httpd. The implementation is then evaluated in chapter 8 with respect to feasibility, serviceability and/or viability. Closing the proceedings part, future research as well as further modifications to the developed concept as well as additional ideas are presented in chapter 9. Concluding the main volume of this thesis, the appendix presents some proofs and further definitions, the milestones set for the interplay of this thesis and [Bom12], as well as a glossary on page 147 defining some additional terms. The appendix concludes with the list of references and auxiliary means starting on page 152. The supplementary (part IV) focuses on selected code snippets, tables, use cases and finally the interface definitions.

The reader shall note: This thesis is composed using British Standard English; hence, words such as 'programme' or 'analyse' are used, rather than 'program' or 'analyze'. Furthermore, some terms are explained directly within the footer of a page as a footnote[1], while others – mostly not so important ones – are briefly explained in the glossary at the end of this assignment paper (refer to page 147). Should there be at least five non-academic sources for data, information, specifications, assertions, declarations or definitions, this thesis will treat them as part of general knowledge and not provide further reference. For all other – to the best of the author's knowledge – not in general knowledge, references are provided in the list of references at the end of this thesis (please refer to page 152). Additionally, it shall be noted that some chapters and/or sections may provide a set of references in their introducing paragraph; hence, only important references will be marked as such. Should a statement not be referenced explicitly within a paragraph, the reader is kindly requested to refer to the references given in the introduction of the chapter and/or section at hand.

Also, it shall be noted that the term 'if and only if' is abbreviated by 'iff', and is used throughout this thesis to shorten the sentences; hence increasing readability.

By times, images/figures will be provided in order to illustrate something. For optimal flow of text, the used typesetting tool (LaTeX) will place these near the point of reference, but this might not always be easily possible; hence the image/figure shall be looked for on one of the next or preceding pages if it is not intuitively near the point of reference.

Short of one final comment, the reader shall be alluded to the fact that each part of this thesis presents its own table of contents on the reverse of the part's front page; hence, it is not necessary to refer to the general table of contents on page 11 when seeking for specific contents of a part.

Finally, the author wishes to point out that this book represents the 'single volume edition' reprint of the original thesis[2]. Therefore, some phrasing may seem odd as the supplementary is explicitly referenced rather than only the corresponding content of the supplementary, only.

[1] For example, this is a footnote.

[2] The original thesis was published in two volumes; the first volume containing parts 1 through 3 and the second volume containing the supplementary part 4.

PART I PROPAEDEUTICS

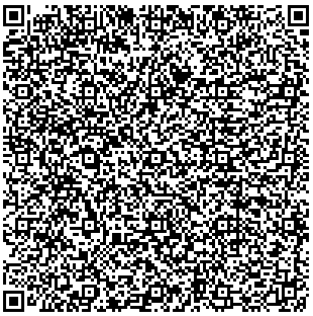
Contents of this Part

Figures within this Part

1 PRELIMINARIES

'All the tissues and organs of the body originate from a microscopic structure (the fertilized ovum), which consists of a soft jelly-like material enclosed in a membrane and containing a vesicle or small spherical body inside which are one or more denser spots. This may be regarded as a complete cell.'

Opening words of Gray's 'Anatomy of the Human Body' (1858), Chapter I, Section 1



PRELIMINARIES

The contents of this preliminary chapter should be known from undergraduate university classes, but they should be briefly repeated in order to ensure that a solid common ground is met when referencing them. Knowledge of such as operators, sets, fields, matrices, the set of natural numbers $\mathbb{N}$ (including 0), relations, functions, et cetera is assumed. The herein defined may not be explicitly recalled in the course of this thesis, but it may proof important when reviewing internals of database operations or data(set) handling.

1.1 MATHEMATIC FUNDAMENTALS

This section will establish the basic mathematic concepts behind array, looping, etc. They will mostly not be referenced to in the written part of this thesis, but they are important for the implementation. It is actually a compilation of definitions and theorems without any further prose interrupting them.

Definition 1.1.1 – Subsets of Natural Numbers

Subsets $\mathbb{N}_{\geq i} \subseteq \mathbb{N}$ contain only numbers recursively derivable from $i \in \mathbb{N}$, meaning all natural numbers larger or equal to i.
The set $\mathbb{N}_{\geq 1}$ is often noted as $\mathbb{N}^+$ whereas $\mathbb{N}_{\geq 0} = \mathbb{N}$.
Finite subsets $\mathbb{N}_{\leq i} \subset \mathbb{N}$ derive from 0 and terminate recursion with i, whereas $\mathbb{N}^+_{\leq i} \subset \mathbb{N}^+$ derive from 1 and terminate recursion with i.

Definition 1.1.2 – Computable Real Numbers

$\mathbb{R}_c \subseteq \mathbb{R}$ is the set of Real Numbers computable by a Turing-Machine.

Definition 1.1.3 – (Sorted) *n*-Tuple

A (sorted) n-tuple has the shape of $t = (t_1, t_2, \ldots, t_n)$. The type of the t_i is irrelevant and does not need to be the same for any t_i and t_j. For sorted n-tuples their order may never be destroyed by swapping them. The elements of t can be extracted or modified using the following operations:

- Extraction of the first element: $t_1 := 1^{\text{st}}(t)$
- Modification of the first element: $1^{\text{st}}(t) :=$ new value
- Extraction of the second element: $t_2 := 2^{\text{nd}}(t)$
- Modification of the second element: $2^{\text{nd}}(t) :=$ new value
- ...

Commonly, a 2-tuple is referred to as *pair*. In more general terms, any n-tuple can be named following the Latin numeral of n: *triple*, *quadruple*, et cetera.

Definition 1.1.4 – Matrix Row Extraction (Column Extraction; Element Extraction)

Let $\boldsymbol{A}$ be a $(i \times j)$-matrix. The matrix row extraction $[]_{\text{row},*}$ (column extraction $[]_{*,\text{column}}$; element extraction $[]_{\text{row,column}}$) produces a row vector (a column vector; an object of the matrix' object type) representing exactly one specified row (column; object) of $\boldsymbol{A}$.

$$[\boldsymbol{A}]_{k,*} = \left[\begin{pmatrix} a_{1,1} & \cdots & a_{1,j} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k_j} \\ \vdots & \ddots & \vdots \\ a_{i,1} & \cdots & a_{i,j} \end{pmatrix}\right]_{k,*} = \begin{pmatrix} a_{k,1} & \cdots & a_{k,j} \end{pmatrix}$$

$$[\boldsymbol{A}]_{*,l} = \left[\begin{pmatrix} a_{1,1} & \cdots & a_{1,l} & \cdots & a_{1,j} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i,1} & \cdots & a_{i,l} & \cdots & a_{i,j} \end{pmatrix}\right]_{*,l} = \begin{pmatrix} a_{1,l} \\ \vdots \\ a_{i,l} \end{pmatrix}$$

$$[\boldsymbol{A}]_{k,l} = \left[\begin{pmatrix} a_{1,1} & \cdots & a_{1,l} & \cdots & a_{1,j} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k,l} & \cdots & a_{k,j} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i,1} & \cdots & a_{i,l} & \cdots & a_{i,j} \end{pmatrix}\right]_{k,l} = a_{k,l}$$

1.2 FURTHER DEFINITIONS

This section continues establishing basic concepts, but focussed more on language theory. As true for the mathematic fundamentals, these formal definitions will not be referenced in the written part of this thesis, but they are also very important for the implementation.

Definition 1.2.1 – Alphabet

An alphabet is a finite nonempty set.

Definition 1.2.2 – Word

Let Σ be an alphabet. Then the set Σ^* of all words over Σ is defined as the set of all finite sequences (*strings*) over Σ and the sequence of length 0 (*empty string*), denoted by ε.
Words are denoted by listing the elements of the sequence.
The length of a word $w = a_1 a_2 \cdots a_n$ with $n \geq 0$ is defined as $\|w\| := n$.

Definition 1.2.3 – Formal Language

Let Σ be an alphabet. A formal language $\boldsymbol{L}$ over Σ is a subset of Σ^*, that is a set of words over Σ.

$$\boldsymbol{L} \subseteq \{w \mid w \in \Sigma^*\} \subseteq \Sigma^* \tag{1.2.1}$$

Definition 1.2.4 – Concatenation

Let Σ be an alphabet and $\boldsymbol{L}_1$, $\boldsymbol{L}_2$ two formal languages over Σ. Then the concatenation-operation $\cdot$ is defined as follows:

$$\boldsymbol{L}_1 \cdot \boldsymbol{L}_2 := \{u \cdot v \mid (u \in \boldsymbol{L}_1) \wedge (v \in \boldsymbol{L}_2)\} \tag{1.2.2}$$

Instead of $u \cdot v$ and $\boldsymbol{L}_1 \cdot \boldsymbol{L}_2$ the valid substitutes uv and $\boldsymbol{L}_1\boldsymbol{L}_2$ may be used.

Definition 1.2.5 – Kleene Star

Let Σ be an alphabet and $\boldsymbol{L}$ a formal language over Σ. Concatenating $\boldsymbol{L}$ with itself the Kleene operator * is defined as follows:

$$\begin{aligned} \boldsymbol{L}^0 &:= \{\varepsilon\} \\ \boldsymbol{L}^{n+1} &:= \boldsymbol{L}^n \cdot \boldsymbol{L} \\ \boldsymbol{L}^* &:= \textstyle\bigcup_{n\geq 0} \boldsymbol{L}^n \end{aligned} \tag{1.2.3}$$

Should $\boldsymbol{L}$ not contain the empty word ε, $\boldsymbol{L} \cdot \boldsymbol{L}^*$ does not contain ε either. A valid notation for $\boldsymbol{L} \cdot \boldsymbol{L}^*$ is:

$$\boldsymbol{L}^+ := \boldsymbol{L} \cdot \boldsymbol{L}^* = \bigcup_{n\geq 1} \boldsymbol{L}^n \tag{1.2.4}$$

Definition 1.2.6 – Regular Expressions

Let Σ be an alphabet. The set of regular expressions over Σ, denoted by Reg_Σ, is the smallest set Reg'_Σ such that:

- $\{\emptyset, \varepsilon\} \subseteq Reg'_\Sigma$,
- $\Sigma \subseteq Reg'_\Sigma$,
- $\forall n \in \mathbb{N}_{\geq 0} : \forall r, s \in Reg'_\Sigma : \ (r + s), (r \cdot s), r^*, r^+, r^n \in Reg'_\Sigma$.

Comment

In descending priority *, $^+$ and ^{n} have the highest priority, followed by $\cdot$. The operator $+$ has the lowest priority. Obeying the so set priorities, parentheses may be omitted. The operator $\cdot$ may be omitted as well (see equation 1.2.2 in definition 1.2.4). For example, the regular expression $\left((x \cdot y) + \left(a^2 \cdot b\right)\right)$ can also be written as $xy + a^2b$.

Comment

Conventional definitions of Reg_Σ do not include the operators $^+$ and ^{n} since they can be equivalently substituted by expressions using the standard operators. However, the additional operators allow more compact expressions, which in return enhance readability for humans.

Definition 1.2.7 – Formal Language of a Regular Expression

Let Σ be an alphabet, $r, s, t \in Reg_\Sigma$, $a \in \Sigma$ and $n \in \mathbb{N}_{\geq 0}$. The formal language defined by the regular expression t, denoted by $\mathrm{L}(t)$, is defined inductively:

- $\mathrm{L}(\emptyset) := \emptyset$,
- $\mathrm{L}(\varepsilon) := \{\varepsilon\}$,
- $\mathrm{L}(a) := \{a\}$,
- $\mathrm{L}(r + s) := \mathrm{L}(r) \cup \mathrm{L}(s)$,
- $\mathrm{L}(r \cdot s) := \mathrm{L}(r) \cdot \mathrm{L}(s)$,
- $\mathrm{L}(r^*) := \mathrm{L}(r)^*$,
- $\mathrm{L}(r^+) := \mathrm{L}(r \cdot r^*)$,
- $\mathrm{L}(r^n) := \mathrm{L}\left(\underbrace{r \cdot r \cdot \ldots \cdot r}_{n \text{ times}}\right)$.

1.3 THE MAPBIQUITOUS PROJECT

The ideas expressed about crowdsourcing later in this thesis (refer to chapter 2 and chapter 6) shall be prototypically implemented in the MapBiquitous project of the Technical University of Dresden (TU Dresden), while developing the generic concepts independent of an actual implementation framework.

MapBiquitous provides the base to develop an indoor location-based service, seamlessly integrating itself into existing outdoor location-based services, such as GoogleMaps. The seamless transition between indoor and outdoor location-based services is imperative to the project and one of the core design concepts in order to maximize user experience. As the project is designed to be a framework for prototypical implementations, future extensions must be supported; hence the concept to be implemented itself must be extensible. So, being developed as a proof of concept implementation, MapBiquitous has been the topic of several assignment papers (Belegarbeiten), theses (Diplomarbeiten) as well as practicals (Komplexpraktika). Hence, detailed information shall be provided in reference work like [Spr11], [Keß11], [Keß12], [Wer12], [GKN12] and [DHP+12], so only an overview shall be provided here. Focus shall be on functional range, architecture and data storage.

1.3.1 History

In the beginning, MapBiquitous was developed to be a application for stationary or mobile personal computers, such as desktops or laptops. Only later it was ported to support mobile phones, in particular smartphones running on the Android operating system. This second version is still being used for a range of student works.

1.3.2 Functional range

The actual goals of MapBiquitous is being a proof of concept implementation for diverse location-based services within building. Hence, MapBiquitous primarily and natively supports the display of building maps as known from map services such as Google Maps or OpenStreetMap. Further, the so displayed building maps are embedded into the Google Maps API. To now, several

buildings of the TU Dresden[3] have been mapped into these building maps, allowing the highlighting of rooms within the buildings. The building maps have been enriched by points of interest ('POI') and rudimental navigation data. These POIs and data may be queried and displayed within the displaying mobile device. As all map related data have been put in context of positioning and MapBiquitous has the capability to determine a users position using GPS (outdoor) and WLAN (indoor), MapBiquitous is able to calculate the presumable position of a user within a building. Extending the positioning to a continuous one, MapBiquitous supports room to room navigation within buildings or from one building to another.

Navigation to a building (not 'into') or to the 'nearest' entrance/exit is desirable, but not yet actually supported. Creation of POIs for entrances/exits is in discussion. Further, navigation per storey is possible, but inter-storey navigation remains desirable [Gru12].

1.3.3 Architecture

MapBiquitous utilises a decentralised client server architecture as displayed in Figure 1.3.1. On the left hand the client components are displayed, whereas the right hand displays these server components. The client components consist of building data stored locally on the mobile device as well as a loader, a renderer and a locator module. These modules have access to the local data and can communicate with each other. The server side consists of a directory service and at least one building server. The directory service provides information upon the responsible building server whenever a client queries building data which is within the responsibility of that server, whereas the building server(s) maintain the actual building data.

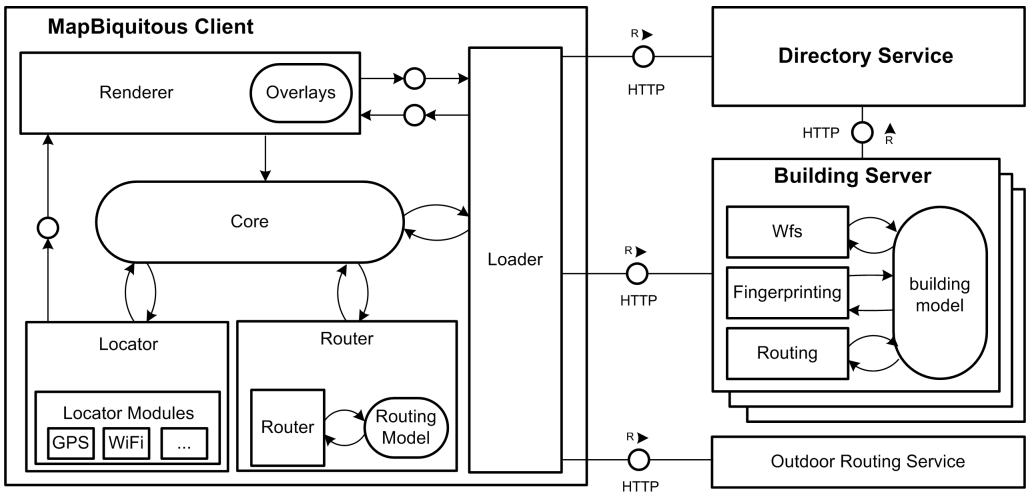


Figure 1.3.1: The basic architecture of MapBiquitous [Spr11]

The renderer module's function is the overlay drawing of the ichnography and floor plans of the diverse buildings, based upon the given building data. The overlay is placed over the map provided by Google Maps, hence the drawn map is an instance of the *Overlay* class of the Google Maps API. Further, the renderer module draws the graphical user interface ('GUI'), the current position as well as some additional data into the overlay. Hence, resulting in a classic mapping and navigation application display.

Current position information is acquired by access to Androids positioning capabilities. The locator module calculates the current position from the provided position information. As mentioned above, outside positioning can be done using GPS as long as four GPS-satellites are within line

[3] Technische Universität Dresden

of sight. Naturally, within building there is no line of sight to GPS-satellites; hence, MapBiquitous relies on WLAN access point data[4]. Determining the position via WLAN is conducted either by triangulation[5] or by fingerprinting[6]. Any position acquired is stored in context of the building data in the local storage of the client and consigned to the renderer module so that it is displayed within the map overlay.

The loader module implements the actual acquisition of building data. Being consigned the data to be loaded by the renderer module, the loader module contacts either the directory service for information on map updates, e.g. after panning/moving the map, or the responsible building server for actual data such as ichnographies, floor plans, WLAN access point data, etc.).

Finally, MapBiquitous contains a navigation module (not in Figure 1.3.1), which calculates routes between two points. Hence, the navigation module has access to the local data storage in order to read building and position data as well as to consign route data to be drawn into the map overlay by the renderer module.

1.3.4 MapBiquitous data – storage and access

The building data used by MapBiquitous are intentionally not stored centralised, but distributed over several building servers which can be accessed via a directory service. This ensures that any building data can be considered 'within the domain' of building representatives[7], allowing decentralised management of the building data. Naturally, the format and structure differ for directory service and building server access. Both have in common the runtime access from the clients, as they can only determine which data are required at runtime.

Any data stored, be it within the client, within a building server or within the directory service, are stored in the World Geodetic System format of 1984 (WGS-84)[8].

Building server

As mentioned earlier, building data are intentionally stored in a distributed fashion on several building servers, each considering a certain building within its responsibility domain. Hence, a building server stores vector-based data for ichnographies and floor plans of buildings as well as position information for WLAN access points within those buildings. Additional semantic data is stored as well, e.g. floor names, room numbers, room occupancy schedules, etc.pp. To efficiently access these data, they need to be stored in a well organised way; hence, a layered structure is used, providing a layer for the ichnography of the building, a separate layer for each storey as well as a separate layer for positioning. The actual vector-based data are represented by polygons; hence, establishing ichnography, storeys, staircases, elevator shafts, hallways and rooms. The additional semantic data are added per polygon. Using the same structure, WLAN access points are stored as polygons with semantic data, including (but not limited to) the MAC address. For accessing the building data, the open source Web Feature Service (WFS)[9] of the OGC is used, as WFS offers several HTTP-based methods for exactly this purpose. MapBiquitous utilises the methods *GetCapabilities()*, *DescribeFeatureType()* and *GetFeature()*. Utilising

[4] Google uses WLAN access point data to enrich outdoor navigation within cities, where there is not always a clear line of sight to four GPS-satellites.

[5] At least three WLAN access points within the range of reception for planar triangulation or at least 6 WLAN access points for steric triangulation.

[6] Signal strengths of WLAN access points near by are being brought into context with other physical measurands.

[7] E.g. the ground keeping unit

[8] WGS-84 comprises a standard coordinate ellipsoid for the Earth (spheroidal reference surface) and a gravitational equipotential geoid that defines the nominal sea level.

[9] `http://www.opengeospatial.org/standards/wfs` – Accessed 7 June 2012

GetCapabilities(), a client can request a list of all available layers of a building. The server response additionally includes the semantic data of each layer. Further information such as type and content of a layer can be requested utilising *DescribeFeatureType()*. Finally, the entire set is requested utilising *GetFeature()*. Nevertheless, the response to either request is always a file in Geography Markup Language (GML)[10] format.

Directory service

Before being able to access a building server, they need to be discovered. Hence, the directory service is utilised to find building servers responsible for a building in which data are required. Following the standard structure of data on a directory server, the building server information are stored on the directory server per available building as a set consisting of the names, the URLs and the coordinates. The stored URL equals the necessary *GetCapabilities()* request of the responsible building server, whereas the stored coordinate is utilised as a search parameter when conducting a building query[11]. The MapBiquitous Client always and automatically sends building queries to the directory service in shape of a XML-based *DirectoryRequest*, containing the geographic limits of the currently displayed map section. Normally, the limits are provided as minimum and maximum longitude as well as minimum and maximum latitude within WGS-84. After receiving such a request, the directory server generates and initiates a database query, resulting in a set of all buildings within the boundaries provided[12]. The set of buildings found is forwarded to the requesting client as a XML-based *DirectoryResponse*.

The MapBiquitous directory service was implemented following the OGC reference implementation precisely. Nevertheless, only a subset of the requests and responses standardised by the OGC was implemented, as not all request/response-pairs were required.

Client

On client side, data are stored within (instantiated[13]) classes. Within each object the different attributes of possible data[14] are stored as in-object attributes, each accessible via corresponding methods. Each building, represented by a *WFSServer* object, contains a generic ArrayList in order to store the different layers of the building in an *WFSLayer* object. These objects at least store their identifier, their name and the layer type, such as ichnography, floor plan, POIs, etc. Furthermore, two generic ArrayList objects are stored for information on rooms and access points. Objects within these lists are of type *BuildingPart* (rooms), containing subobjects for name, usage and other attributes, or of type *WLANAccessPoint* (access points), whose subobjects additionally contain the MAC address of the access point. Geographic vector data of rooms and access points are stored in a subobject as an instance of *GeoObject* of the JavaGIS API. Any instantiated *WFSServer* object is managed by the *LocationModelAPI* class. Access methods (get and set) to the *EFSServer* and *WFSLayer* objects are implemented in this class.

Simplified, the client is divided into four distinct areas: A *Locator* module that provides positioning services, a *Loader* module that acquires data from servers, a *Renderer* module providing visualisation of map and positioning data, and binding those three together, a central information base providing all the current data. The simplified structure is depicted in Figure 1.3.2.

[10] GML is an instantiated derivate of XML and is used to transmit location-dependent data. It is standardised within OGC standards.
[11] Especially when considering the last known GPS coordinate as 'entrance' to the building in question...
[12] I.e. all buildings within the displayed map overlay will be found.
[13] E.g. a building is stored as an instance of the *WFSServer* class.
[14] E.g. the name or the URL to the responsible building server within *WFSServer* or identifier, name and typ of a layer within *WFSLayer*.

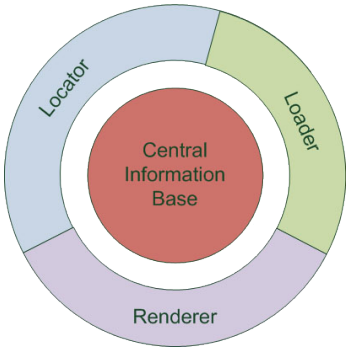


Figure 1.3.2: The simplified structure of MapBiquitous' client [GKN12]

Building data are loaded and converted into client usable data by the *Loader* module. Independent of where the request is sent to, either directory service or building server, several objects are instantiated at runtime, generating representations of the *Directory*, *GetCapabilities* or a *GetFeatures* requests, which are handled asynchronously as *AsyncLOD1Task*, *AsyncLOD2Task* or *AsyncLOD3Task* runtime objects, whereas each possible level of detail (*LOD* 1, 2 or 3). An *AsyncLOD1Task* represents a request to the directory service, generating a new *WFSServer* object with the information provided by the directory service. A LOD deeper, an *AsyncLOD2Task* represents the *GetCapabilities* request to a building server, generating new *WFSLayer* objects, which are enlisted into the generic ArrayList of the generating *WFSServer* object. Additionally, using a *GetFeature* request to the building server, an *AsyncLOD2Task* also loads a building's ichnography. Finally, an *AsyncLOD3Task* represents the *GetFeature* request to a building server, loading the vector, meta and access point data into the *WFSLayer* objects, storing them into the ArrayLists for rooms and access points. While LOD1 and LOD2 loading tasks are automatically initiated by the client, the actual loading process of LOD3 is initiated by the system user as soon as the user desires the GUI to display a storey not currently present in the client's data. The internals of the *Loader* module as well as the schematic loading intents/notifications between the *Loader* and *Renderer* modules are displayed in figures 1.3.3 to 1.3.5.

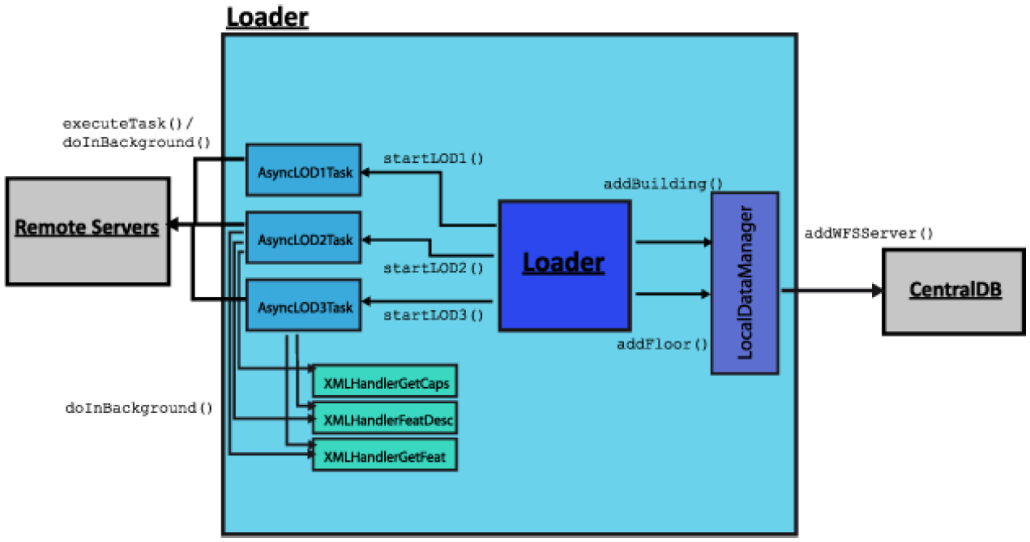


Figure 1.3.3: The basic architecture of MapBiquitous' *Loader* module [Keß12]

Within the structure of the Android API, the actual command for the *Loader* module to retrieve new data from either the directory service or a build server is issued by the *Renderer* module. When the user is either moving the map overlay or explicitly wanting to display a different storey, the *Renderer* module sends a message[15] to the *Loader* module with an instruction of

[15] An 'Android Intent3'

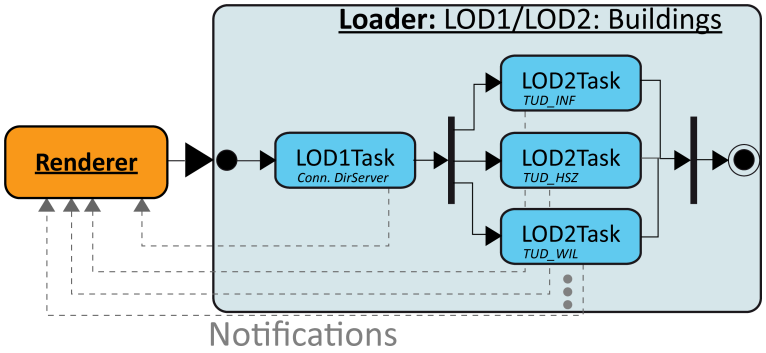


Figure 1.3.4: The basic intents/notifications communicated between MapBiquitous' *Loader* and *Renderer* modules when loading LOD1 and LOD2 data [GKN12]

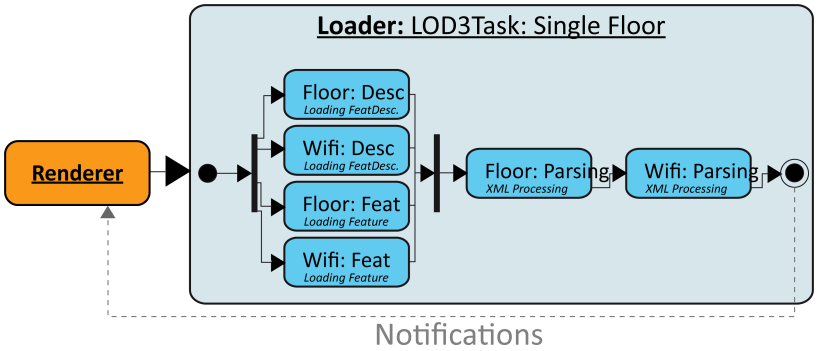


Figure 1.3.5: The basic intents/notifications communicated between MapBiquitous' *Loader* and *Renderer* modules when loading LOD3 data [GKN12]

what to load. After successfully loading the requested data, the *Loader* module will send a response message to the *Renderer* module with the loaded data, initiating a re-rendering of the displayed map overlay by the *Renderer* module. The basic message-driven communication of the *Renderer* and *Loader* modules is depicted in Figure 1.3.6.

1.3.5 Navigation

According to [Keß12], MapBiquitous currently offers a proof of concept implementation of building-spanning navigation. Unfortunately, an actually working navigation could not be reproduced for building-spanning routes during the creation of this thesis, while indoor-navigation could be reproduced for a limited set of routes. Therefore, it shall be assumed that the concept might work with a corresponding simple bugfix; nevertheless, neither the author of this thesis, nor the author of the concomitantly written assignment paper [Bom12] were able to actually fix the issue...

Anyway, the concept envisages the navigation to be a graph-solving problem utilizing Dijkstra's algorithm[16]. The graph at hand is built by connecting POIs considered being *between* the two destinations. The POIs are considered the vertices of the graph, whereas the connections are being considered the edges. The determination whether a POI is *between* the destinations is conducted with help of symbolic coordinates, which primarily describe the distance between the destinations. Only POIs considered in the vicinity[17] of the probable route are introduced into Dijkstra's algorithm. For this to work, the symbolic coordinates includes semantic data as well

[16] Refer to Theorem D.0.1 in the supplementary.
[17] Using symbolic coordinates

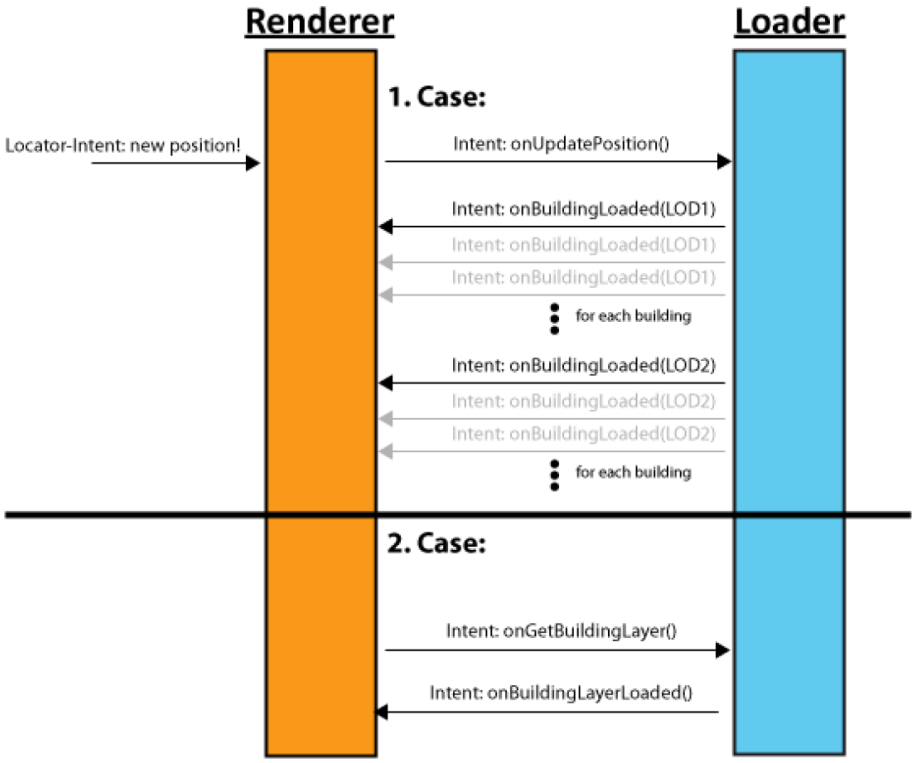


Figure 1.3.6: The basic message-driven communication MapBiquitous' *Renderer* and *Loader* modules [Keß12]

as location data. I.e. positions in symbolic coordinates include information on in which building and on which storey the POI is located at. Further information on distances to other symbolic coordinates as well as room data are included in WGS-84 standard coordinates. The latter are used to interconnect symbolic coordinates when navigating between two buildings. The actual outdoor navigation is then done by existing services, such as Google Maps. Using the semantic data on buildings and storeys, the indoor navigation may be limited to the building at hand, while maintaining a route within a storey, only using allowed waypoints, such as staircases or entries, to change storeys or buildings.

2 CROWDSOURCING

'You can't rule the world in hiding. You've got to come out on the balcony sometimes and wave a tentacle.'

The Fourth Doctor (Thomas Stewart Baker)

CROWDSOURCING

Crowdsourcing presents itself to be a rather young concept in the field of computer science, emerging in the late 1990s or early 2000s respectively. For example, the *SETI@home* project was released to the public on 17 May 1999, making the concept of a distributed calculation known to the public. As crowdsourcing is such a young concept, currently there exists no commonly agreed upon definition of what crowdsourcing actually is. There exists a vague consensus on basic ideas, so intuitively most people agree on what the ought to believe crowdsourcing is, but no agreeable formal definition has been accepted in general. Even though, crowdsourcing can be divided into two types: explicit crowdsourcing and implicit crowdsourcing; once again, not clearly defining what both of them are. Nevertheless, this chapter shall introduce an attempt at giving a definition of crowdsourcing as well as a taxonomy of crowdsourcing. Different derivates shall be introduced and categorised according to the defined taxonomy. Finally, a context to the similar term 'social sensing' shall be established.

2.1 ATTEMPT AT A DEFINITION

Closely following Howe, Estellés-Arolas and González-Ladrón-de-Guevara [EG12, How06] in context of Quinn and Benderson [QB11] a common consensus on the definition of crowdsourcing shall be established. Even though it is hard to find a consensus as definitions vary broadly, it is imperative to agree on the common ground in order to build up an usable concept of crowdsourcing, especially a taxonomy of explicit crowdsourcing versus implicit crowdsourcing.

Comment

Sometimes the equivalent term 'crowd sourcing' can be found in sources. The author of this thesis attempts to consequentially use 'crowdsourcing'.

Currently, there is no generally accepted definition of crowdsourcing, only a large number of widely varying definitions. In order to have a starting point, Definition 2.1.1 [EG12] shall be given, which – according to self-made statements – Estellés-Arolas and González-Ladrón-de-Guevara came up with after studying 40 definitions of crowdsourcing.

Definition 2.1.1 – Crowdsourcing

Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken.

Thus, Crowdsourcing is a distributed online or offline process involving a flexible group of contributors (the 'crowd') and tasks to be (partially) completed by the contributors. The flexible group of contributors is not well defined and is subject to dynamic changes in composition, structure and taskability. Hence, crowdsourcing can be categorised as a distributed problem-solving model with different algorithms. The entity making use of the crowd and providing the task(s) is considered the 'crowdsourcer'.

In most concepts – as is true for this thesis – the crowd is composed of humans which are at the same time users of a system which itself is the crowdsourcer. Often, the solutions submitted to the crowdsourcer are composed only of evaluable/analysable data, which are thereafter owned by the crowdsourcer, making the crowd take the role of service providers.

Often, either the crowd as a total or individual contributors are compensated monetarily[18], with prizes[19] or benefits[20], or with recognition[21]. In other cases, there is no reward as the crowd might not even know they are participating in the crowdsourcing process, e.g. Facebook Inc. gathers information on persons who are not even members of the community by simply analysing any data community members upload and putting them into relation; most of the Facebook users are not aware that they contribute personal data of dégagé thirds, such as telephone or address data, even though those thirds might give their consent to such contribution. The crowdsourcers generally benefit by large numbers of solutions/information being provided inexpensively or even for free; this of course, only if considered a crowd large enough.

2.2 TAXONOMY OF CROWDSOURCING

Different derivates of crowdsourcing include, but are not limited to:

- *Crowdvoting* is the concept of a website gathering votes on a certain topic, e.g. which picture provided by a community shall be awarded 'picture of the year'.
- *Crowdwisdom* is the concept of collecting vast quantities of information, organising this information and deducing a commonly agreeable recollection of the picture behind the information. Wikipedia can be considered the example for crowdwisdom, as users contribute new information, update existing information or delete information considered irrelevant or false. As the crowd has different backgrounds[22], the commonly agreed recollection can be considered broad and well founded.
- *Crowdfunding* is the concept of collecting vast financial resources by involving a crowd out of which each contributor only contributes a small monetary support. In general, but not as a rule, the goal of the fund-raiser is clearly defined. Kickstarter may be the example for crowdfunding as it is the biggest website for funding creative projects, having raised over 100 million USD. The resources contributed are only allocated to the crowdsourcer if the defined goal is reached, otherwise the resources are returned to the crowd.
- *Crowdpurchasing* is the concept of leveraging the collective purchasing power of the crowd in order to attain a reasonable discount or even the best price a dealer or manufacturer is willing to offer in general. Letsbuyit.com is an example for this concept, where users are presented with several levels of achievable discounts, depending on the size of the crowd involved.
- *Crowdworking* is the concept of distributing parts of a dividable task to the crowd. In general, but not as a rule, the tasks at hand are considered challenging for a computer, but easy for a human. In 2006, the National Aeronautics and Space Agency asked people to review images returned by the Stardust project on NASA's website[23] in order to find images with 'interesting' dirt.

[18] Test subjects are often monetarily compensated in the field of pharmaceutical research.

[19] Enquiries often offer the possibility of participating in a competition after submitting query-forms, promising equal chances of winning prizes to all participants.

[20] Dropbox Inc. offered users participating in the beta-test of their automated photo-upload function additional 500MB of online-storage, which would remain even after the beta-test concluded, for each 500MB of photos uploaded. `http://forums.dropbox.com/topic.php?id=53104` – Accessed 1 June 2012

[21] Search for Extra Terrestrial Intelligence at home (SETI@home) offers their users the possibility to download and print out a certificate of appreciation for certain levels of credit points. `http://setiathome.berkeley.edu/cert_print.php` – Accessed 1 June 2012

[22] E.g. there might be veterinarians, truck-drivers, police force, etc. involved in maintaining an article about roadkills.

[23] `http://stardust.jpl.nasa.gov/home/index.html` – Accessed 1 June 2012

A more comprehensive look at some of the derivates can be found in chapter 2.

Independent of the derivates, there exist two concepts of crowdsourcing: explicit crowdsourcing and implicit crowdsourcing. The definition of what 'explicit' and what 'implicit' are do vary, making a general definition almost impossible. The two most commonly agreed distinguations of the two are:

- Explicit crowdsourcing lets users cooperate and actively contribute to the crowdsourcing process, while implicit crowdsourcing means that users solve a problem as a side effect of something else they are doing, and
- Explicit crowdsourcing gives users awareness of the crowdsourcing process and lets them decide when to participate in the crowd sourcing process, while implicit crowdsourcing keeps the crowdsourcing process transparent to the users, requiring no user interaction at all.

As for the MapBiquitous project and the creation of this thesis and G. Bombach's assignment paper [Bom12], both distinguation concepts shall be united, maintaining the agreed concepts of implicit crowdsourcing. Clearly, both distinguations follow independent differentiators: the immediacy of the crowdsourcing process on the one hand and the crowd's awareness of the crowdsourcing process on the other hand. As both differentiators are orthogonal to each other, the following taxonomy shall be proposed:

Definition 2.2.1 – Taxonomy of Crowdsourcing

There exist four types of crowdsourcing which can be placed within a taxonomy build on awareness and immediacy. Awareness is the concept of the crowd being aware of the crowdsourcing process taking place, whereas immediacy is the concept of acquiring required information either directly (strong correlation between collected data and derived information) or indirectly (loose correlation between collected data and derived/computed information). Hence, there are four types of crowdsourcing:

- **Aware Direct Crowdsourcing** (*ADC*)
  Aware crowdsourcing directly correlated to the data required.
- **Unaware Direct Crowdsourcing** (*UDC*)
  Unaware crowdsourcing directly correlated to the data required,.
- **Aware Indirect Crowdsourcing** (*AIC*)
  Aware crowdsourcing loosely correlated to the data required.
- **Unaware Indirect Crowdsourcing** (*UIC*)
  Unaware crowdsourcing loosely correlated to the data required.

The following conventions shall be valid: As *aware direct crowdsourcing* is explicit in awareness as well as immediacy, this type of crowdsourcing shall be labelled 'explicit crowdsourcing' within this thesis, whereas *unaware indirect crowdsourcing* is implicit in awareness as well as immediacy, so it shall be labelled 'implicit crowdsourcing' within this thesis.

The entire taxonomy as defined is depicted in Figure 2.2.2. Above introduced abbreviations (ADC, UDC, AIC, UIC) originate in the differentiators explicitness or implicitness, for which the first letter represents the awareness differentiator, and the second letter represents the immediacy differentiator.

Independent of the distinguation between implicit and explicit crowdsourcing, commonly agreed upon (and hence often used), implicit crowdsourcing can take two forms. These forms of implicit crowdsourcing cannot clearly be mapped on either differentiator, but shall be mentioned here anyway, as there exists an obvious tendency towards the awareness differentiator, so – even though not fitting exactly – they can be mapped accordingly.

- Standalone implicit crowdsourcing allows the crowd to contribute as a side effect of the task they are actually solving or of the usage of the crowdsourced system, whereas

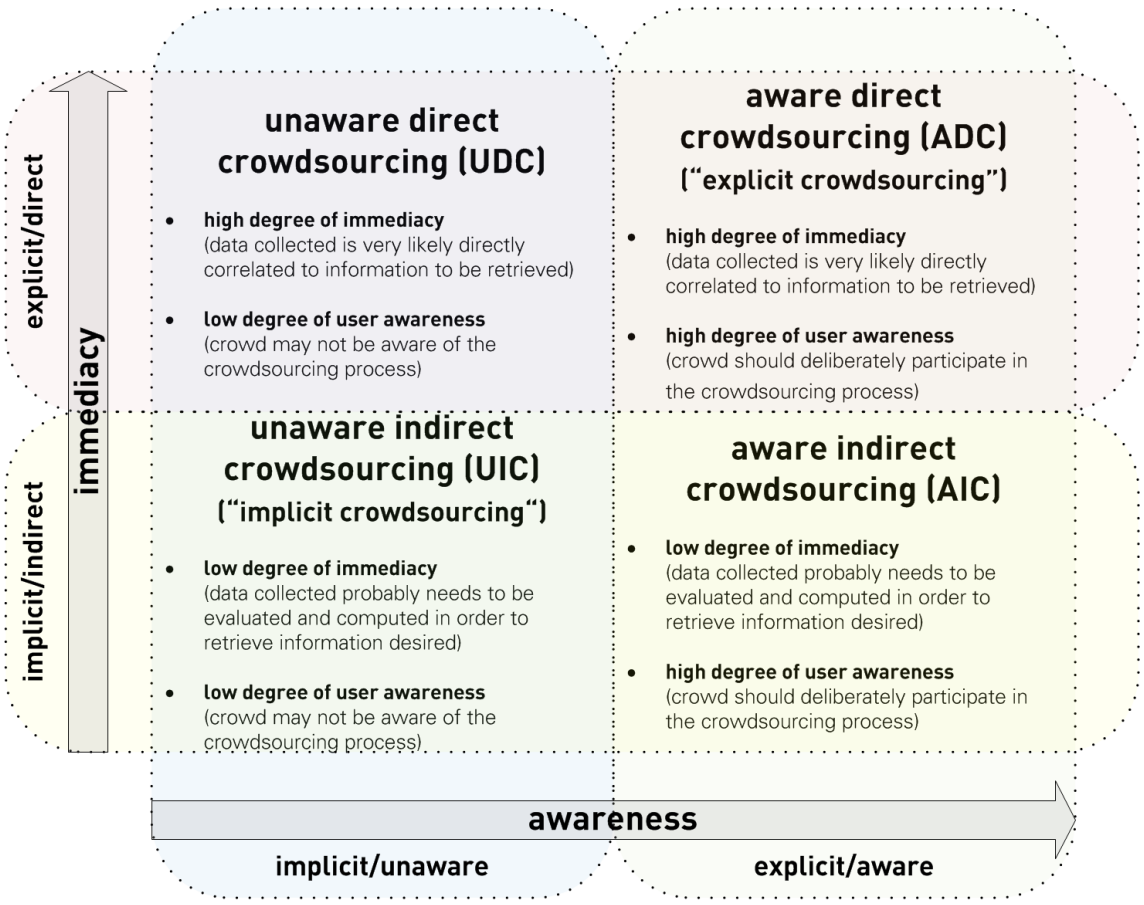


Figure 2.2.2: The taxonomy of crowdsourcing as to be used within this thesis

- Piggyback implicit crowdsourcing takes information deducted from the crowd's general feedback and acquires data from them[24].

The tendency to the context of awareness is obvious, but it is not clear enough to justify naming *unaware direct crowdsourcing* 'standalone crowdsourcing' and *unaware indirect crowdsourcing* 'piggyback crowdsourcing' within the taxonomy.

Inexpediently, the four types of crowdsourcing may not always be clearly distinguished, making the crowdsourcing process present itself as a hybrid, which shall simply be called 'hybrid crowdsourcing'.

Corollary 2.2.3 – Refinement of the crowdsourcing taxonomy

The taxonomy of crowdsourcing implicitly includes hybrids and hence may be refine by adding information upon the awareness and immediacy:

- **Unaware Crowdsourcing**
  The hybrid of UDC and UIC.
- **Aware Crowdsourcing**
  The hybrid of ADC and ACI.
- **Direct Crowdsourcing**
  The hybrid of UDC and ADC.
- **Indirect Crowdsourcing**
  The hybrid of UIC and AID.
- **(Entirely) Hybrid Crowdsourcing**
  The hybrid of all four types of (non-hybrid) crowdsourcing.

The refined taxonomy enriched by hybrids is depicted in Figure 2.2.4.

Corollary 2.2.5 – Further refinement of the crowdsourcing taxonomy

The refined taxonomy of crowdsourcing includes further hybrids beyond the hybrids introduced in Corollary 2.2.2 ('mostly unaware slightly indirect hybrid crowdsourcing', 'slightly unaware mostly indirect hybrid crowdsourcing', etc.).

It shall be noted, that data protection issues (refer to section 5.1) may arise. Hence, users should be asked whether they wish to contribute collected data, or not. This would automatically make the entire crowdsourcing process aware to the crowd, so the concept of awareness should be refined.

Definition 2.2.6 – Crowd Awareness

Awareness within a crowdsourcing process arises iff the crowd is regularly confronted with the fact that crowdsourcing is taking place.

In the sense of Definition 2.2.4, the crowd is unaware of the crowdsourcing process if they are asked for permission once at the beginning of the crowdsourcing process. This is very important when considering that humans tend to not read the fine-print when using software, etc.

With the so defined taxonomy, the above mentioned derivates can be exemplarily categorised into the taxonomy types ADC (explicit crowd~) and UIC (implicit crowd~), without limiting to either of them.

[24] Google stores a cookie on users' computer while surfing to Google's search engine. When later surfing to affiliated partner websites, the cookie can be used to trace users and deduct behavioural patterns, which in turn can be used to optimise advertisement cashback.

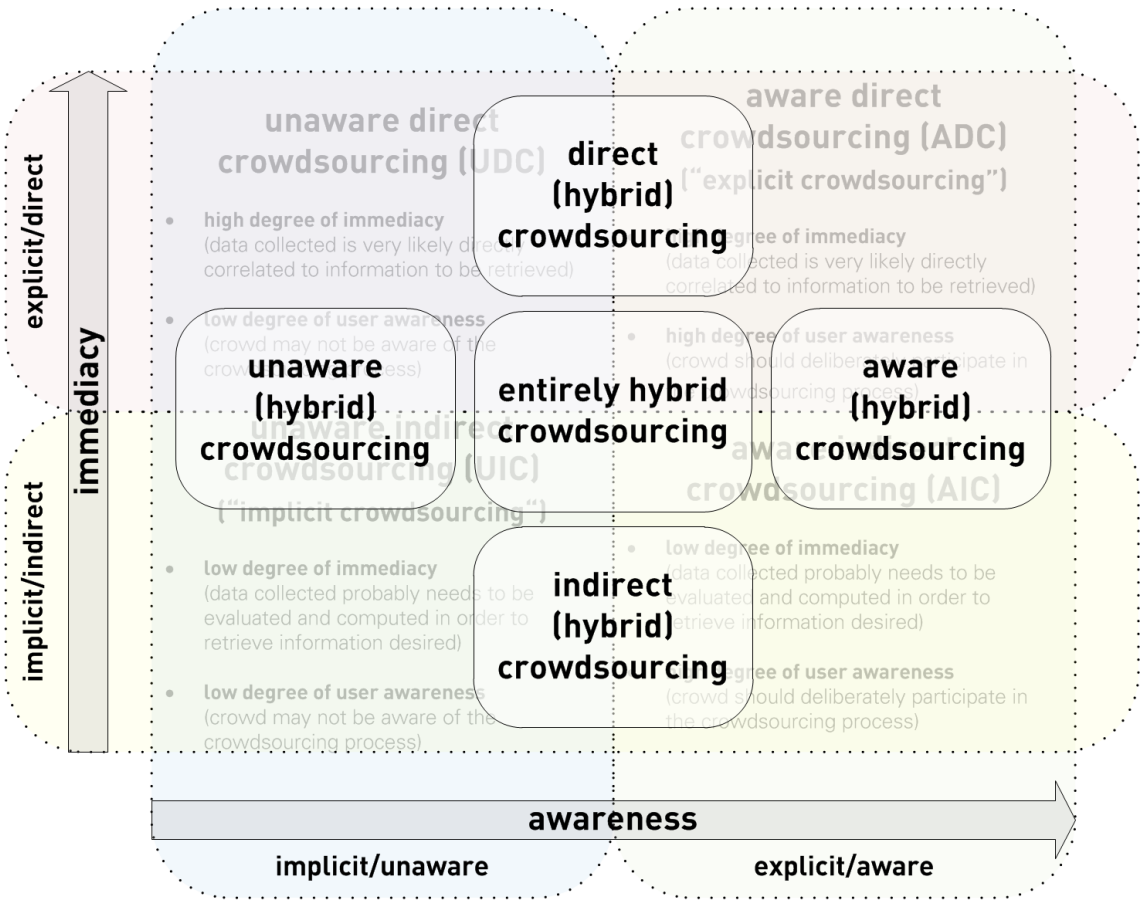


Figure 2.2.4: The taxonomy of crowdsourcing refined with hybrids

- **Crowdvoting**:
  - *Explicit crowdvoting*: Being presented a selection of choices, users pick their favourite. – The crowdsourcer is the designer of the selection trying to grasp the general opinion of a set of users, whereas the crowd is the set of users wanting to know, what thoughts others have on the topic in question.
  - *Implicit crowdvoting*: Analysing the sales of products, a shop can create a ranking of their products. – The crowdsourcer is the shop wanting to sell high quantities of favourable products, whereas the crowd is the total set of customers simply buying the products.
- **Crowdwisdom**:
  - *Explicit crowdwisdom*[25]: In light of the annular solar eclipse that took place on 20 May 2012[26], astronomers all over Japan observed the border of the annulus in order to find the exact[27] line of annular observability. With the information collected the exact circumference of the sun at that time could be calculated. – The crowdsourcers were the scientific community of Japan wanting to determine the exact circumference of the sun at that time, whereas the crowd was the set of all observers sending in information on annular observability.
  - *Implicit crowdwisdom*: Hardware manufacturer often offer a cashback system, awarding end-users with financial benefits for returned hardware at the end of the hardware's lifecycle. – This is of course an example for explicit crowdsourcing when considering only the cashback, but considering the manufacturer as crowdsourcer wanting to acquire knowledge upon hardware quality or durability, the customers sending in their hardware can be considered the crowd implicitly providing the knowledge on quality or durability.
- **Crowdfunding**:
  - *Explicit crowdfunding*: A start-up wanting to start production of a product may call for supporters, offering them a reduced special price when the product is finally available. – The crowdsourcer is the start-up requiring funding support, whereas the customers are the crowd providing the funds, expecting the reduced price as an award.
  - *Implicit crowdfunding*: Users regularly visit a website with advertisements placed on it. – The crowdsourcer (explicit and implicit) is the website's owner receiving money for each advertisement displayed and/or clicked, whereas the explicit crowd is the set of users clicking the advertisements (creating payable click impressions), and the implicit crowd is the set of users visiting the website and having the advertisement only displayed in their browser (creating payable view impressions).
- **Crowdpurchasing**:
  - *Explicit crowdpurchasing*: A group of users may form an aggregation of purchasers in order to achieve the optimal price from a manufacturer. – The crowdsourcer is the manufacturer demanding to sell large quantities of their products, whereas the crowd is the aggregation of purchasers wanting to reach the price minimum.
  - *Implicit crowdpurchasing*: A supermarket sells articles at a fixed price, but buys them at varying prices. – The crowdsourcer is the supermarket willing to win the lowest purchase price, the crowd is the set of all customers buying the article from the supermarket. Depending on sells, the order amount varies; hence a purchase price fluctuation may occur. The supermarket may support this by advertising the article in order to push sells.
- **Crowdworking**:
  - *Explicit crowdworking*: An academic work is put to public discussion. – The crowdsourcer is the author of the work asking for feedback, whereas the crowd is the set of all readers reading and commenting on the work.
  - *Implicit crowdworking*: People are invited to test a new hiking course for free. – The crowdsourcer is the owner of the hiking course, whereas the explicit crowd is the set of people using the hiking course for free, while the implicit crowd is the same set of people compacting the grounds and foundations of the newly created hiking course.

Once again, it shall be mentioned that a more comprehensive look at some of the derivates can be found in chapter 2.

[25] Information on this example provided by an article of the NHK World News on 25 May 2012.
[26] UTC; 21 May 2012 in Asian time; eclipse 58 of Saros-cycle 128
[27] In some areas in Japan, observation units were set up every 180 metres.

2.3 SOCIAL SENSING

An approach at gathering information focussed on humans and their behaviour is 'social sensing'. The sole sources of information gathered are the human being and its surroundings. Humans themselves as monitors [ASS+10] gathering strongly correlated measurands as well as the humans' social interactions as monitors gathering loosely correlated measurands [MCLP10, Tel07] allow information retrieval in a social context. The approach is well distributed and crowd-based, so it is fair to assume it to be at least a derivate of crowdsourcing, which – as a reminder – is not limited to only humans.

Looking at the was social sensing is described to work in general (e.g. in [ASS+10, MCLP10, Tel07]), a striking similarity to crowdsourcing can be observed. A party is interested in information, which shall be provided by observed (and distributed) humans. This model is an exact match to the crowdfunder/crowd-relation of crowdsourcing (refer to Figure 2.3.1).

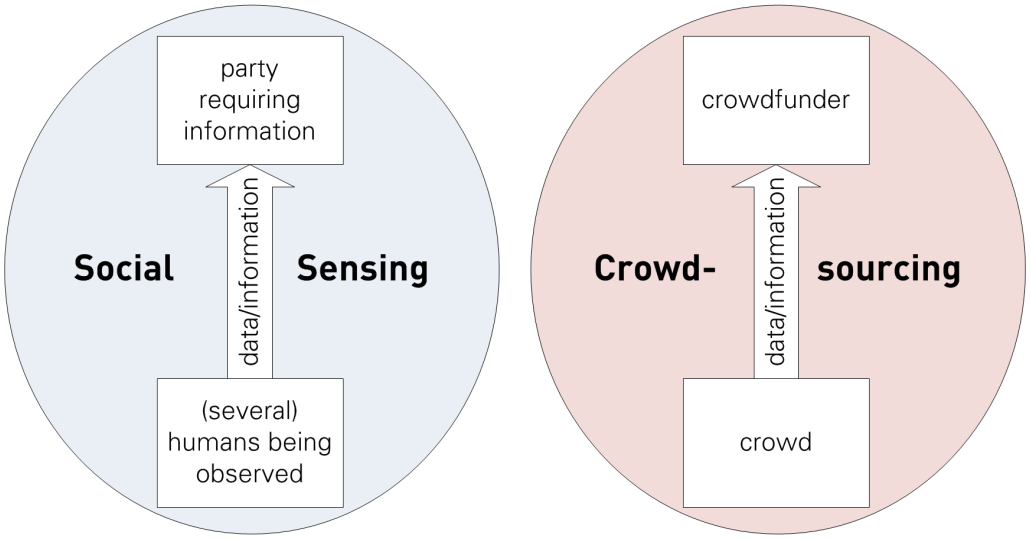


Figure 2.3.1: Comparison of social sensing and crowdsourcing; striking similarity

Further, the same taxonomy as introduced in Definition 2.2.1 can be applied accordingly. On the one hand, when social sensing takes place, it is possible to have the humans either be aware of the social sensing, or not be aware of it. Hence, following Definition 2.2.1, social sensing should also be able to be typified along the awareness differentiator, leading to *aware* and *unaware* social sensing. On the other hand, data acquired by social sensing can be strongly correlated to the information required[28], but also weakly correlated to the information required[29]. Hence, also the immediacy differentiator as introduced in Definition 2.2.1 can be applied, leading to *direct* and *indirect* social sensing.

Taking all the similarities into consideration, it should be fair to proclaim that social sensing is in fact crowdsourcing, but limited to humans and data/information strongly correlated to humans and their social surrounding.

Comment

For the remainder of this thesis, it shall be assumed that the made proclamation is valid. Nevertheless, limiting the scope to social sensing would be unjust; hence, the broader crowdsourcing will be used in chapter 6.

[28] E.g. the required information could be (in absolute numbers) an answer to the question, 'How many female friends does a man aged 30 have in average?'

[29] E.g. the required information could be an answer to the question, 'Does a larger circle of friends reduce the risk of allergic coryza?'

3 EXISTING CROWDSOURCING CONCEPTS

'Don't blink. Don't even blink! Blink and you are dead! They are fast, faster than you could believe. Don't turn your back, don't look away, and don't blink. Good luck!'

The Tenth Doctor (David Tennant) in 'Blink' (BBC, 2007)

EXISTING CROWDSOURCING CONCEPTS

Comment

> *The contents of this chapter were contributed by Gerd Bombach and are not original work of the author of this thesis, who only contributed the translation from German into English. These contents focus on crowdsourcing applications and concepts, which are imperative to know as they describe the motivation for research in this field. On a first glimpse, they may seem to not be very related to the contents of this thesis; however, it is important to have a well founded motivation for something (in this case the conceiving of a crowdsourcing architecture) rather than 'just doing it'. Therefore, this chapter shall help identify the motivation.*

The concept of utilising crowdsourcing to collect and enhance map data has been in use by several project for a few years. The most prominent example can be considered to be *OpenStreetMap* which has been developed at the College University London. Utilising a steadily growing crowd of supporters, especially in well developed areas map data of high quality could be aggregated within a very short time. According to [NZZ11], the aggregated data provides better information than comparable solutions of commercial tenderers. However, this is exactly the problem with this approach as the quality of the map data is proportional to the population density and the development state of the area in question. While highly developed countries exhibit nearly perfect map data, poorly developed areas manifest large uncovered mapping areas.

Especially the field of map data generation and correction for indoor applications is currently under intensive investigation. Currently, the main problem is an exact, reliable and cheap positioning technology for these indoor applications. Further reference on the problems is provided in [Gru12].

Nevertheless, also the creation and enhancement of map data utilising the capabilities of the crowd requires further investigation, even though several applications of this field have emerged into practical use.

The remainder of this chapter shall introduce some scientific ansatzes, followed by three existing commercial ansatzes that create or enhance their map data by crowdsourcing.

3.1 SCIENTIFIC ANSATZES

3.1.1 Geowiki: Creation of Outdoor Maps utilising Crowdsourcing

In edition 11/2011 of 'Computer' [Mas11] different geowikis are presented. The most prominent example once again is OpenStreetMap. As a minor project, designed to fulfil the demand of cyclists, *Cyclopath* is introduced. The results of the presentation clearly show up that commercial ansatzes are only interested in data able to generate a profit in the future. Map data for cyclists do not count in this category in the moment. Even further, cyclists require more exact and highly adaptable map data for their navigation, which is only possible by unprofitable expenses for centrally organised map data providers. Utilising crowdsourcing, this effort can be distributed to several motivated supporters.

Another factor interceding for crowdsourcing are times of crisis. While centralised instances often collapse in crises, the crowd may still provide map data. In the aftermath of the earthquake in Haiti in 2009, maps based on topical satellite images were created within a very short time. These maps were essential for rescue forces in order to advance into the devastated areas.

Additionally, it was possible to visualise the extend of damages by comparing before and after images. [All12]

In [Mas11] also the problems with visualising the ascertained data is described. A solution would be – following OpenStreetMap's ansatz – to utilise a domain specific view. OpenCycleMap displays information for cyclists, while ÖpnvKarte emphasises information for the local public transport[30]. However, both views share the same map data base.

An evaluation in [PMT10] on the bicycle route requests via maps amended by the crowd concludes that calculated routes could be shortened by an average of 1 km after the crowd had added shortcuts and catenations of roads via parking lots.

In summary, generation and correction of maps utilising a large crowd has superior quantity and quality compared to non-crowdsourced map data.

3.1.2 Motivation of the Crowd

An important question to answer with respect to crowdsourcing is about how to motivate people to participate. Different ansatzes were already discussed in chapter 2. [PMT10] has conducted a field study with 1500 users of Cyclopath and identified an elect core of a few users actively contributing, while the majority of users limit their usage of the system to passive functions. They then analysed ansatzes to motivate more users to become active participants. Therefore, within the field test users had to amend map data, in particular whether intersections were actually intersections, or bridges spanning over roads. The researchers concluded that visual highlighting of areas with limited data encouraged users to actively participate; with respect to the field study, intersections with few data were highlighted. They even asserted that users once they had started to contribute data would actively continue as long as visual highlights were given – some users even contributed for areas that were not highlighted.

However, another hypothesis could not be verified: The researchers assumed that users would contribute substantially more for areas they knew rather than unknown areas. The result show, only by visual highlighting users could be animated to (also) contribute to areas they knew.

In summary, [PMT10] concluded that visual highlighting of demanded data is imperative for active participation of the crowd. Monetary stimuli are not necessarily required.

[KSV11] asserted similar results from a survey among the participants of Amazon's 'Mechanical Turk'. The mechanical turk is a crowdsourcing marketplace where crowdfunders can offer tasks that can be worked on by the users of the platform. Fields of work covered inter alia range from image recognition, language transcription, product description to identification of performers of music. Mechanical turk focuses on monetary reward; however, for some type of tasks these reward were as minimal as a few cents. Therefore, [KSV11] analyses why certain tasks are favoured over others. A model including intrinsic[31] and extrinsic[32] motivation was developed. A corresponding survey concluded that the monetary compensation is prime motivation for the survey participants. However, intrinsic motivation follows closely.

Summarising, [KSV11] concludes that monetary compensation is essential if crowdsourcing participants are not able personally benefit intrinsically from their participation. Additionally, tasks must be designed to not be monotonic, allow the participants to recognise the value of their contribution and allow a certain degree of freedom in the processing of the task.

[30] 'ÖPNV' is a German acronym for 'Öffentlicher Personennahverkehr', which literarily translates to 'Public Person Local Transport'.
[31] E.g. variety of the tasks, identification with the task, feedback after completing a task, etc.
[32] E.g. monetary compensation, improvement of skills, unlocking of confined tasks, etc.

3.1.3 GSM Measurements as an unaware indirect Crowdsourcing Method

Ascertainment of GSM cell site locations and their signal strengths at selected positions may improve positioning within a building significantly. For this, GSM information could be automatically ascertained by users of an indoor navigation application while they are using the system. This can be considered an unaware indirect crowdsourcing (UIC) derivate. Determining a position on basis of several different sources of information may help to improve the positioning significantly by applying a particle filter.

Engaging GSM-based positioning, [OV05] concluded an advantage of GSM by broader availability of the infrastructure. Theoretically, anywhere GSM cell site locations are available, positioning should be possible without having have to deploy additional hardware. The results are very promising and actually prove good positioning based on GSM information. However, this approach requires availability of information of several GSM cell site locations in range – i.e. triangulation must be possible. And this is the crux of the matter, as most of the GSM hardware built into smartphones does not allow access to GSM information other than of the cell site location the device is currently connected to. This is especially a problem for smartphones operating under the Android operating system.

3.2 COMMERCIAL ANSATZES

3.2.1 Google Indoor Maps

With 'Indoor Maps' Google introduced an extension to their map service at the end of 2011. Google determines a person's position by triangulating them with WLAN access point information. For this to work, building ichnographies and positions of access points within the build must be available beforehand. In the early phase of provision, Google themselves supplied these information for a few large malls and airports in the United States of America. However, in April 2012 Google introduced the crowdsourcing app 'Google Maps Floor Plan Marker'[33]. Using this app, users of the indoor navigation service are supposed to contribute map data on their own. For this, users need to select their building of choice beforehand using a special website[34] and upload an ichnography, e.g. by scanning an evacuation map, etc. Afterwards, the ichnography must be placed and rotated within a satellite image until a perfect match is reached. The so provided ichnography is available in the Android app afterwards. The user is then supposed to stride up and down the building following a calculated path, covering all areas of the building while ascertaining WLAN access point information.

Summarising, Google Indoor Maps is exemplar for aware direct crowdsourcing (ADC). Currently[35] indoor maps exist for selected building in France, the United Kingdom, Switzerland, Japan and the United States of America[36]. In Germany, one can upload ichnographies; however, neither can the Android app be downloaded nor can one use the app[37].

[33] `https://play.google.com/store/apps/details?id=com.google.android.apps.insight.surveyor` – Accessed 18 October 2012
[34] `https://maps.google.com/help/maps/floorplans/` – Accessed 18 October 2012
[35] 18 October 2012
[36] A comprehensive list is available at `http://goo.gl/ZfNJO` – Accessed 18 October 2012
[37] Assuming one obtains the app from 'somewhere'.

3.2.2 Unaware Crowdsourcing: Traffic Congestion Prediction

By now, the concept of floating car data (FCD) is being used by many companies as it is several years old, well researched and established. Serial-production sees FCD present since 1999.

In short FCD amends the cars with modules collecting data while the cars move in the traffic. Mostly, the data is more or less directly ascertained from the cars GPS-hardware. The data is then (pre)processed and transmitted to a central processing station via GSM or UMTS connection. As soon as a certain critical mass is reached, real-time traffic monitoring and congestion prediction becomes possible. This information is then sent back to the FCD modules via the same internet connection used earlier. The cars' onboard navigation units can then calculate alternative routes avoiding the existing and/or possible congestions.

As [Gro] describes, BMW has extended this concept by not only ascertaining position and speed of the vehicles, but by also ascertaining further sensor data. Exemplary, data from ABS[38], ESC[39] and temperature sensors can allow estimations on the possibility of black ice. These additional data are sent to the FCD centre together with the other data. In return, the FCD centre is able to warn other vehicle following on the same path or operating in the vicinity. As these data sent are ascertained by the vehicles anyway, the extension of the FCD system to use these data is very cost saving. The entire ascertainment is a background process, totally unaware to the operator of the vehicle. At most, the owner/operator of the vehicle is asked for permission to ascertain and transmit these data when powering up the FCD system for the first time.

In [STW02] the possibilities to build a traffic information system based only on FCD data are discussed. Several hundred vehicles of a taxi business were equipped with FCD modules, transmitting the vehicles' positions once per minute. [STW02] concluded that the ascertained data was sufficient to reproduce the entire city's traffic volume in real-time and allow dynamically optimised routing.

Currently, similar concepts are intensively in use. However, since smartphones are available to a broad mass of consumers, the availability and density of data ascertainers has grown large. Google, Apple as well as TomTom in cooperation with Vodafone anonymously ascertain and process the positions and speeds of all vehicles/persons utilising their products[40]. Explicit navigational use of the products is not even required, as the products constantly ascertain these data in the background and transmit them to the corresponding servers. In this spirit, it is sufficient to turn on a product and carry it with oneself in order to contribute to the (crowdsourced) data ascertainment.

3.3 CONCLUSION

In summary, one can perceive a clear importance of crowdsourcing in context of map data today. Many well working applications are available that would not function without the contributing crowd. By now, also the users are used to investing work into a service that does not seem to immediately benefit them. By dividing large tasks into smaller micro-tasks, projects can be established that commercial centralised products can not. The best example for this is Wikipedia. Further demand of research clearly exists, not limited to the field of computer science.

[38] Anti-lock Breaking System
[39] Electronic Stability Control
[40] The wording is intentionally 'product', as the vendors provide own and licensed hardware as well as own and licensed software.

4 RELATED WORK

'Work is the refuge of people who have nothing better to do.'

Oscar Wilde

RELATED WORK

For any academic work, especially for a thesis, it is very important to narrow down the scope by pointing out related work. This proves rather simple for any crowdsourcing project itself, as chapter 2 demonstrates. Unfortunately, examination is limited to the scope of the crowdsourcing projects and their general functionality, as it seems to be impossible to find current information on the actual implementation as soon as crowdsourced data reaches the crowdfunder.

This is especially true for commercial products, e.g. Google's 'Street View'. Users of Street View are able to contribute user placed information tokens, such as a panorama image or restaurant critique; basically this can be considered crowdsourcing of (crowd) knowledge. However, it is impossible to gather information upon how the data is stored once it reaches Google's servers[41]. Due to the competing products of rival companies, information is treated very secretive. The best data available from Google is a guide[42] for their 'Google Earth Server', an Apache httpd derivate, which delivers map data to accessing clients. However, it is not described how Google themselves use this server, especially in context of crowdsourced amendment/correction of data.

In a strange sense, the same problems also apply for 'open' projects, such as OpenStreetMaps, which has a rather comprehensive Wiki[43]; however, no information on the server infrastructure is available. There also exists a list[44] of indoor navigation projects using OpenStreetMaps as a basis, but they all share the lack of server infrastructure information. The list includes, but is not limited to:

- BA Indoor Routing Web-App by Andreas Hubel
- The indoorOSM project from OSM data
- FootPath by RWTH Aachen
- The Leadme project by Richard Atterer

Another prominent representative of an 'open' project with comprehensive information on the application concept, however not on the server architecture, is OpenRoomMap[45]. It is developed and used for extensive research at the University of Cambridge; therefore, an extensive list of publications exists[46]. However, to best knowledge of the author of this thesis, none of the listed 199 publications focuses on the server architecture of OpenRoomMap.

Due to the difficulties mentioned above, this important section of related work culminates in the realisation that there is no current related work presentable for the server side of crowdsourcing. The more it is important to present a good concept (refer to chapter 6) and actually share the information, as done within this thesis. However, there exists one rather old crowdsourcing project that actually shares information: the Berkeley Open Infrastructure for Network Computing (BOINC).

---

[41] The author of this thesis isn't even sure whether Google uses several servers (might be a fair assumption), or only one server.
[42] To be found at `http://www.google.com/enterprise/earthmaps/earth_server.html` – Accessed 17 October 2012
[43] It is available at `http://wiki.openstreetmap.org/wiki/Main_Page` – Accessed 17 October 2012
[44] The list is to be found at `http://wiki.openstreetmap.org/wiki/Indoor_Mapping` – Accessed 17 October 2012
[45] `http://www.cl.cam.ac.uk/research/dtg/openroommap/` – Accessed 18 October 2012
[46] Available at `http://www.cl.cam.ac.uk/research/dtg/www/publications/` – Accessed 18 October 2012

4.1 BOINC – BERKELEY OPEN INFRASTRUCTURE FOR NETWORK COMPUTING

The Berkeley Open Infrastructure for Network Computing (BOINC) project started in February 2002 and its first usable version was released 10 April 2002. After a phase of crowdknowlege – i.e. open programming – the project was deployed under the GNU General Public License on 18 November 2003. However, actual operations did not commence until 9 June 2004 when Predictor@home[47], the first project to use the BOINC architecture, was launched. The basic idea of the BOINC project is to procure a reliable platform for distributed computing projects by means of the BOINC architecture; hence, the project can be considered a PaaS[48]-derivate. This PaaS-aspect is what makes BOINC interesting for this thesis and justifies its status as related work, as the goal of the thesis is to develop a reliable architecture for crowdsourcing. Several crowdfunders – i.e. the maintainers of the buildings – want to crowdsource information ascertainment. What type of information does not matter, as different crowdfunders may be interested in different information. Comparing this to the BOINC architecture, a striking similarity is obvious, as different crowdfunders – i.e. the computing projects – want to crowdsource information processing; what kind of information does not matter, as the different projects are interested in different types of processing. Hence, basic concepts and ideas of the BOINC architecture should be applicable to the concept to be developed in this thesis. This is especially true for the HTTP-based communication as well as the server components.

Basically, the BOINC architecture separates crowdsourcers and crowdfunders from each other, as to be expected from any crowdsourcing architecture. More into detail, the separation subdivides into several components which shall be briefly introduced.

- **Participant's side (client-side)** The participant's side of the BOINC architecture roughly follows the MVC[49]-pattern.
  - *Core Client*
    A command line programme running as a background-process on the participant's computer, monitoring and controlling all installed project applications pursuant to the participant's specifications. It buffers work units and handles communication with the schedulers and data servers of the installed projects.
    The core client does not conduct any crowdsourcing computation, i.e. it is not part of the distributed computation. It can be considered the controller of the client-side.
    This component does not need to be project-individual as its operation does not depend on the project installed on the client.
  - *BOINC Manager*
    A graphical interface for the configuration and monitoring of the core client. It relies on the core client as controller; hence, it can be considered a view.
    This component does not need to be project-individual as its operation does not depend on the projects installed on the client.
  - *Boinc Command Line Interface*
    A command line interface for the configuration and monitoring of the core client. It relies on the core client as controller; hence, it can be considered a view.
    This component does not need to be project-individual as its operation does not depend on the projects installed on the client.
  - *Project Applications*
    The actual application logic of each installed project. They are downloaded and deployed into the computer's processing queue by the core client and compute the work units provided by the core client. Project applications are monitored by the core client via shared memory access and can therefore be considered the models.
    This component has to be project-individual as it facilitates the project operation on the client.

[47] Predictor@home was a distributed computing project at the University of Michigan, established by the Scripps Research Institute in order to predict protein structures from protein sequences and to test, evaluate and enhance algorithms for structure-prediction of known and unknown proteins.
[48] Platform-as-a-Service
[49] The Model-View-Controller is a widely used programming design-pattern.

- **Server-side**
  Each project must implement its own server infrastructure based on a web server, PHP, Perl or ASP as script language, and a MySQL database. The server components must be deployed at least once, but multiplication for redundancy or load balancing is explicitly encouraged.
  - *Scheduler*
    A CGI[50]-programme on the project's web server. It assigns work units to the clients redundantly, i.e. several clients are assigned the same work unit. After computation of the work units has completed the scheduler has to be provided with a success or failure information, but not the actual results. All activities are logged into the local MySQL database.
    This component does not need to be project-individual as its operation does not depend on the project running it.
  - *Data Server*
    This is a simple web server only supporting HTTP communication. All clients are expected to retrieve their assigned work units from this server and to upload their results to this server.
    This component does not need to be project-individual as its operation does not depend on the project running it.
  - *MySQL Database*
    This component facilitates storage of status and processing information. It does not store the work units or results; they are stored in the file system directly.
    This component does not need to be project-individual as its operation does not depend on the project running it.
  - *Validator*
    A project-specific programme that revises and validates the clients' results. In general, validation is enforced by comparison of the redundant results of several clients for the same work unit; hence, the validator does not need to compute the work units itself. Validated results are pushed to the assimilator.
    This component has to be project-individual as its operation depends on the project running it.
  - *Assimilator*
    A project-specific programme accepting only validated work unit results. It dresses the result data for further scientific or academic analysis. Often, these results are stored into another MySQL database.
    This component has to be project-individual as its operation depends on the project running it.
  - *File Deleter*
    After assimilation of the result the corresponding work units and results are deleted by this component in order to free then unnecessarily occupied memory.
    This component does not need to be project-individual as its operation does not depend on the project running it.
  - *Transitioner*
    Implementing a virtual multi-core pipeline, this component monitors the distributed computation. It receives work unit states from the scheduler by sharing access to the same MySQL database and regularly checking it. As soon as sufficient results have been uploaded to the data server, the transitioner activates the validator for further processing of the results.
    This component does not need to be project-individual as its operation does not depend on the project running it.

For communication the components use the standard hypertext transport protocol (HTTP). An extension to HTTP is the data server protocol, a XML-derivate-based packet-oriented communication, used for work-unit-result uploads. The data server protocol is designed to prevent denial of service attacks on the data servers. Independent of the protocol used, all communication is bound to port 80, allowing BOINC communication through common firewall configurations.

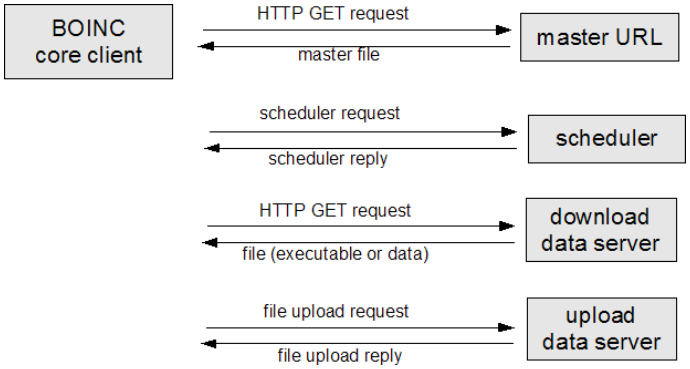


Figure 4.1.1: Basic communication within the BOINC architecture. (© University of California, Berkeley – published under the GNU Free Document License 1.2)

According to the BOINC project, communication follows[51]: With the corresponding original description from the BOINC project:

- 'The client downloads the page from project's master URL. From XML tags embedded in this page, it obtains a list of domain names of schedulers.'
- 'The client exchanges request and reply messages with a scheduling server. The reply message contains, among other things, descriptions of work to be performed, and lists of URLs of the input and output files of that work.'
- 'The client downloads files (application programs and data files) from one or more download data servers. This uses standard HTTP GET requests, perhaps with Range commands to resume incomplete transfers.'
- 'After the computation is complete, the client uploads the result files. This uses a BOINC-specific protocol that protects against DOS attacks on data servers.'
- 'The client then contacts a scheduling server again, reporting the completed work and requesting more work.'

The internals of the data server protocol shall be omitted here as they can be found in the BOINC Wiki[52].

An exemplary architecture with clients, servers and communication paths is depicted in Figure 4.1.2.

[50] Common Gateway Interface is a standard for data exchange between a web server and other programmes running on the same machine.
[51] `http://boinc.berkeley.edu/trac/wiki/CommIntro` – Accessed 7 October 2012
[52] `http://boinc.berkeley.edu/trac/wiki/FileUpload` – Accessed 7 October 2012

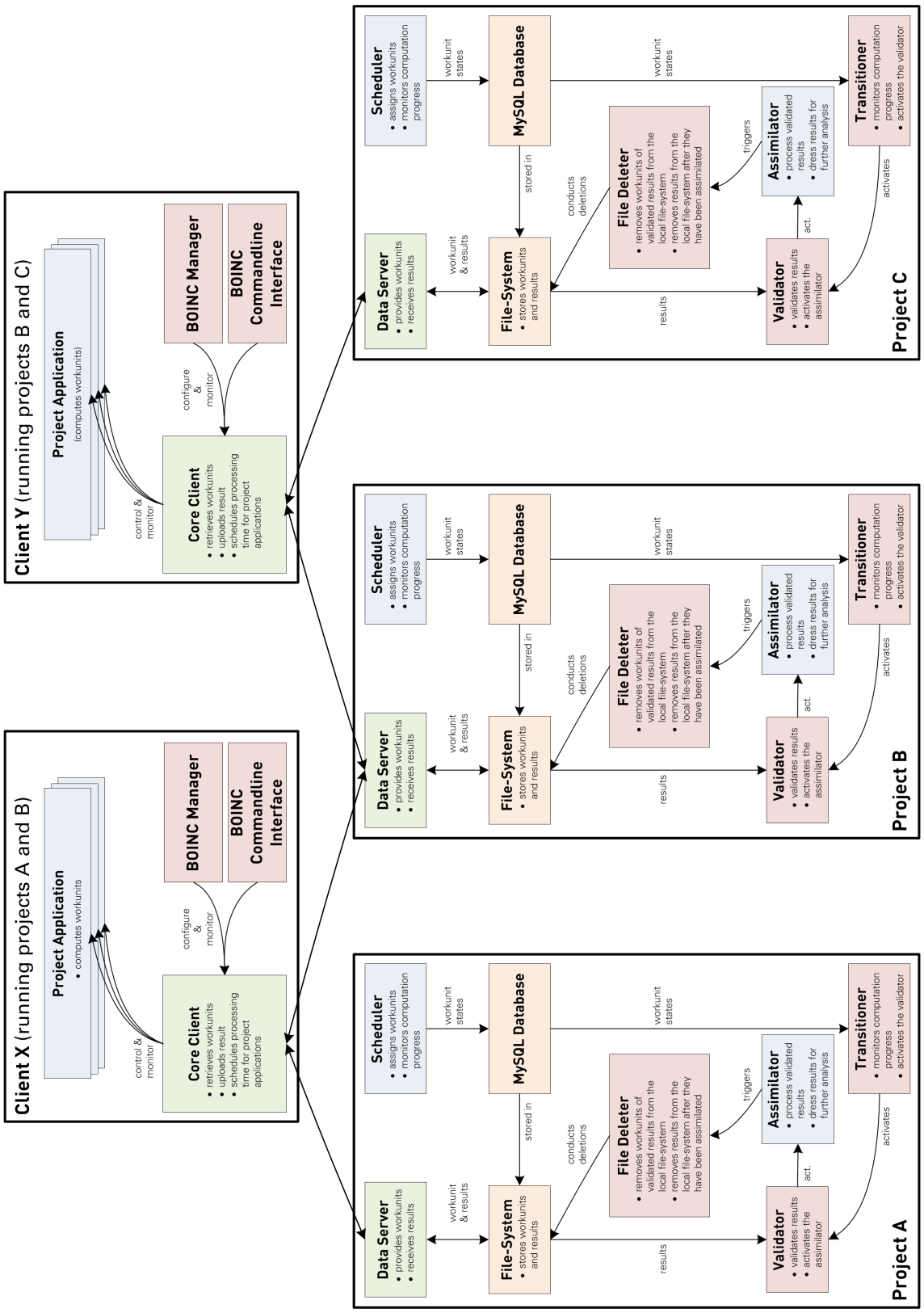


Figure 4.1.2: An example for a project distribution with the BOINC architecture.

PART II PROCEEDINGS

Contents of this Part

Figures within this Part

Tables within this Part

5 REQUIREMENTS REVIEW

'A strange game. The only winning move is not to play. How about a nice game of chess?'

W.O.P.R. (War Operation Plan Response), WarGames, 1983

REQUIREMENTS REVIEW

As the goal of both Gerd Bombach's assignment paper as well as this thesis is the introduction of crowdsourcing to the MapBiquitous project, a well thought-out concept must be envisaged. But, before such a concept can be actually designed, it is necessary to determine what the concept is supposed to cover in means of not only functionality, but also data to be processed, etc. In short, a requirements review is to be conducted.

Gerd Bombach and the author of this thesis have identified a set of design goals for the crowdsourcing to be introduced which shall be adopted as requirements. Starting from the tasks handed out to both authors[53], a rather limited set of two requirements can be directly derived. The set includes automated processing of crowdsourced data, and limitation to a few selected data to collect without obstructing future addition of further collectible data. – Taking a closer look at these two requirements, a rather complex subset of requirements can be identified.

Firstly, the requirements subset definitely include a demand to keep the system working even if a user should decide not to participate in the crowdsourcing. This first requirement automatically mates with another requirement derived from the original set: Users shall be given the choice whether to participate in the crowdsourcing or not. For this to work, any existing interfaces must remain untouched or – if inevitable – should be only slightly modified. For the case of users declining participation in the crowdsourcing, the subset review terminates here, while further requirements can be identified for the case of participating users.

After having conceived the crowdsourcing taxonomy in Definition 2.2.1, the authors wish to implement exemplary use cases for all four main types of crowdsourcing in the crowdsourcing process itself, but due to limited time, this wish cannot be regarded as a requirement. In lieu thereof, the awareness differentiator shall be of primary focus, making examples for ADC and AIC a requirement. Maintaining crowd awareness (refer to Definition 2.2.4) automatically mates with this requirement. U∗C-crowdsourcing shall be prepared, but not implemented.

Independent of the type of crowdsourcing, the authors wish to add another requirement, which is neither specified in the tasks, nor derivable from the task: anonymity shall be emphasised. Any user participating in the crowdsourcing shall – if possible – not be made known to the crowdfunders, in this case the building server. To the very least, their personal data (their identity) shall remain disclosed from the crowdfunders. Further, the consent to participate in the crowdsourcing should be revocable so that a user can decide to return to be a user using the existing default interfaces of the system. This requirement mates with the demand to be able to delete/remove/erase/revoke ('drer') submissions from the system, which cannot be fully fulfilled, as this requirement collides with another: at some time the crowdfunders wish to sift submissions and rely on them being available thereafter; hence, submissions should only be *drer*able for a certain time. In correlation, an overview and possibility of retroactive modification of past submissions is desirable for users. Also, legal as well as business management aspects may arise, including the demand to be able to reproduce distinct users' action, e.g. in order to exclude trolls[54] and other negative influences from the system. This demand pairs with a need for quality review of some sort, e.g. by holding back information until it has been verified by a certain number of contributors.

In order to provide at least two different examples of crowdsourcing, the authors wish to actually implement correction of positions provided by the MapBiquitous system (i.e. fingerprint data) and additionally prepare GSM information extraction. For this, GSM data should be ascertained and stored, but not processed. Future additions to the set of desired crowdsourcing

[53] The task at hand for this thesis can be found on page 3; Gerd Bombach's task is almost identical, but with focus on explicit crowdsourcing.

[54] For example, a user constantly submitting false data in order to annoy/aggravate other users of the system may be considered a troll.

methods should not be obstructed; therefore, modularisation of the application logic should be considered, allowing easy addition of further crowdsourcing methods.

As a last requirement towards system functionality, for reasons of usability, a direct feedback of crowdsourcing submissions is desirable for the users of the system.

After having focussed on functionality, other aspects of requirements should be investigated. Of course, it is in the nature of the dynamic size of the crowd that it is unforeseeable how many or how few participants decide to contribute to the crowdsourcing process. Hence, it is logical to demand that the architecture to be designed provides service for few as well as many participants. In general terms, the system needs to scale well, meaning good scalability should be a design goal. While having it scale well, the system should be of good performance, too. Normally, performance can be archived by optimisation of code, communication protocols, et cetera, but this is not applicable for the academic approach presented within this thesis and [Bom12]. Especially the source code should be expandable for following generations of students working on the project, so the focus should be set on good documentation as well as easily understandable and expandable code, rather than optimisation. Further, the system itself should be able to support addition of further crowdsourced data and/or application logic. Hence, the system should ideally be modularised, allowing easy modification and addition of application logic. The application logic should then not be limited to processing of stored data, but it should also allow the configuration of ascertainment and storing parameters. In this spirit, each module should define which data types and communication expectations it has.

Looking even further into the systematic aspect of the architecture requirements, security once again hits the eye. Not only should security be investigated as described earlier on functional level, but also in means of encryption of communication channels, encryption of stored data, hardness against attacks, etc. From this, a basic set of requirements can be directly and easily be derived: communication and data storage shall be encrypted, and access requires authentification and authorisation. But, these two requirements must be met without binding users of the system to certain components of the architecture; hence, a final requirement can be identified: users of the system shall be loosely bound to infrastructure components.

5.1 DATA PROTECTION ISSUES

Originating in the nature of crowdsourcing, a huge amount of data is transmitted from the crowd to the crowdfunder. In the context of the MapBiquitous project, large quantities of data are uploaded from the clients and evaluated on the servers. Possibly, this happens without the users' awareness of such processes. Hence, it is imperative to ensure a high degree of data protection, especially the users' privacy must be protected at all costs, without leaving authorities out of scope. A review of legal boundaries of collectible data, collected data, protectable data and protected data is required.

The review of data protection issues is imperative in light of recent events. As stated in [Bec12, Sch12], on 4 September 2012 the hacker group AntiSec released files with over a million Unique Device identifiers (UDID) of devices running on the iOS operating system. Of course, the exemplary implementation of the concepts to be introduced in this thesis will be conducted under the Android operating system, but the problem is true for Android, too. Basically, many programmers tend to use the UDID as an identification means to separate app users from each other. When obtaining the UDID, it is theoretically possible to determine all apps a user has downloaded from an app store. Even further, as [Bec12, Sch12] summarise, some weakly protected services divulge private information such as email addresses or usernames for social networks such as Facebook. In lieu thereof, goal 2d turns out to be a valid addition by the authors.

Comment

The review of legal boundaries shall be limited to a focus on Germany, as this thesis is written at a German university, the MapBiquitous project is located at a German university, and the scope is within the field of computer science, not (international) law...

5.1.1 Legal Boundaries in the Federal Republic of Germany

In general, Germany has a strict policy on data protection for government institutions, such as the police, registry offices, etc. Further, physician, dentists schools and universities are subject to strict requirements of data protection, as well. As the MapBiquitous project is currently under development at a German university, German data protection requirements apply automatically and compulsory. Additionally, any government or non-government organisation are subject to the German 'Bundesdatenschutzgesetz'[55] (BDSG) [Bun90], which basically defines a minimum consensus of data protection standards. Other boundaries, such as compulsory data storage and allowed data storage are defined in Germany's 'Telekommunikationsgesetz' (TKG)[56] (TKG) [Bun04] and 'Telemediengesetz'[57] (TMG) [Bun07]. Nevertheless, fundamental rights, such as informational self-determination, strictly apply due to the German constitution ('Grundgesetz' (GG) [Bun49]).

The rights and obligations defined within the BDSG cannot be voided, only complemented by additional, stricter rules and obligations, or they must be explicitly waived by the affected person or organisation. Many organisations hence define corresponding rules in their terms of use or end user license agreements. For example, Facebook Inc. explicitly asks their users to waive their rights in Germany, stating the place of jurisdiction to be the Republic of Ireland. This 'export' of data protection jurisdiction is only permitted within the European Union (EU) and European Economic Area (EEA) as long as the organisation at hand has no branch in Germany. As soon as a branch is opened in Germany or the organisation only resides outside the EU/EEA, German data protection laws automatically apply for individuals in Germany.

The BDSG defines:

- section 1 (§1 through §11): general and shared regulations,
- section 2 (§12 through §26): data processing by public authorities,
- section 3 (§27 through §38a): data processing by private institutions,
- section 4 (§39 through §42): special prescriptions,
- section 5 (§43 through §44): penalties and administrative fines, and lastly
- section 6 (§45 through §46): temporary arrangements[58].

Fundamental concept of the law is to protect individuals from being harmed in their personal rights. Hence, the law basically prohibits ascertainment[59], processing[60] and use[61] of personal data, unless the affected individual explicitly gives their consent. Personal data therein is defined as any data which can be used to describe personal or factual relations of an individual. This data must not contain the name, as individuals can be determined using other data such as telephone numbers, email addresses, personnel numbers, etc., as well. In contrast to personal data, anonymous data can be considered any data indeterminable, such as a calculated, irreversible SHA hash of the personnel number. Pseudonyms are considered personal data, not anonymous data. Neither personal data, nor anonymous data and therefore not scope of the law are corporate bodies and their data.

[55] Federal Law on Data Protection
[56] (Federal) Law on Telecommunication
[57] (Federal) Law on Telemedia
[58] The Bundesdatenschutzgesetz was last amended in 2009.
[59] Acquisition of data on individuals directly from the individual or from third parties with knowledge of the data.
[60] Storing, modification, transmission, blocking and/or deletion of data.
[61] Any handling of data which can neither be considered ascertainment nor processing.

Basically, ascertainment, processing and use of personal data is prohibited, unless a law allows them or explicit consent (§13(2) et sqq. BDSG) is given. As soon as a law allows ascertainment, processing or usage, or an individual gives their explicit consent, the law dictates(§3a BDSG) eschewal and thrift methods. Especially, data should be anonymised whenever possible, or they should not be stored in the first placed if possible.

Any non-government organisation must appoint a data protection official as soon as at least ten (§4f(1)4 BDSG) employees regularly work with personal data using automated methods[62], or twenty (§4f(1)3 BDSG) employees regularly work with personal data using manual methods[63]. Should no data protection official be appointed, automated methods are due notification (§4c(1) BDSG) to controlling authorities.

Rights defined for affected individuals are unwaivable (§6(1) BDSG) and include:

- disclosure if and upon which personal data of the individual are stored,
- disclosure of the source of the personal data of the individual,
- disclosure of the intent of the ascertainment, processing and use of the personal data of the individual,
- summons to correct false or falsified personal data of the individual by the individual,
- summons to delete personal data of the individual by the individual,
- summons to block[64] personal data of the individual by the individual, and
- appeal at the responsible controlling authority by the individual.

Disclosure is to be granted free of charge (§19(7) BDSG) by authorities and government organisations, whereas private organisations may charge (§34 BDSG). The rights of disclosure upon which personal data are stored and their origin/source may be denied iff public interest outweigh the individual's personal rights, e.g. when government secrecy applies. Denials must be investigated on a per-individual basis and may not be declared a general rule; further, denials and their grounds must be put into protocol with the responsible controlling authority.

A totally different aspect of German laws takes scope not on data protection, but on compulsory data collection. Currently, only telcos are obliged to mandatory storage of connection data, such as IP addresses, access times, etc. by European directives. But, these directives have not sufficiently been implemented into German law. Hence, laws (e.g. §113 TKG) implement telcos' obligations to share connection data with (police) authorities, but whether the actual sharing is reconcilable with the German constitution is in dispute. In fact, fundamental rights declared in the German constitution such as the secrecy of telecommunications (Art. 10 GG) and the right of informational self-determination (Art. 2, Sect. 1 GG in conjunction with Art.1, Sect. 1 GG) seem to be incommensurate with the wanted sharing of communication data. The Federal Constitutional Court of Germany has ruled that German telecommunication laws must be amended in order to commensurate to the constitutional rights [Bun12]. According to laws as by time of the ruling of the Federal Constitutional Court of Germany, telcos must delete communication data immediately after connections are terminated, unless the data is required for accounting purposes. Hence, especially IP data are currently deleted after 7 days. Beyond that, data storage on servers is limited, as well. Data may only be stored to an extend required to offer services requested; any storage beyond that is illegit (§15 TMG). There exists a variety of corresponding court rulings, e.g. from the State Court of Berlin [Lan07].

Summarising, the concept to be developed for crowdsourced data ascertainment, processing and evaluation in chapter 6 must adhere to the legal boundaries at any time, especially the user of the system must be made aware of the ascertainment and processing. Therefore, the user should be presented a legal disclaimer before being able to participate in the crowdsourcing process. Further, the user must be given the possibility to revoke their consent at any time.

[62] E.g. computer databases

[63] E.g. paper record cards

[64] I.e. personal data may not be transmitted to third parties; should they have been transmitted already, any third party to which the personal data have been transmitted is to be informed of the block, and the third party must delete or block, as well.

5.1.2 Collectible Data

Practically, any information transmitted to the servers can be considered collectible, even the transmission itself can be considered information[65]. On a more obvious level, any information digitisable can be considered collectible. Hence, the following list[66] shall point out data collectible by MapBiquitous clients; any collectible data shall also be considered available not only in current time, but also available for past times due to storing/caching on the client:

- accelerometer/gyroscope,
- barometer,
- battery (load level, temperature, current, voltage, . . .)
- Bluetooth (ID, devices in reception range, PAN[67]-properties, ...)
- compass (direction of north pole, magnetic field strength, . . .),
- connected devices (headset, external antenna, remote control[68], pay device[69], . . .),
- GPS position (calculated results of GPS raw data),
- GPS raw data (received satellites, satellite time, satellite positions, . . .),
- GSM data (cell-ID, cell time, distance to connected cell tower, . . .),
- hardware (chipset, CPU, display resolution, hardware-IDs, firmware data, . . .),
- microphone (on/off, recordable acoustic measurands, . . .),
- NFC (ID, devices in reception range, . . .),
- radio (channels in reception range, current channel, antenna used, . . .),
- SIM (IMEI, contacts on SIM, SMS on SIM, . . .),
- state (screen on/off, . . .),
- statistics (conversations, SMS, calendar, temperature development, charge level development, . . .)
- stored data (raw data, hashes, SMS, email, pictures, . . .),
- temperature (CPU, chipset, device casing, . . .),
- time (local time, boot/on time of device, cellular time, time of contacted NTP server, . . .),
- WLAN (on/off, connected/disconnected, connected access point, access points in range, data rate, encryption used, . . .),
- . . .

From these obvious information, a wide range of derivable information can be gathered, e.g.:

- preferences (surfing, on/off times, brand, etc.),
- safeness of usage (battery preferred being full versus always using the device until it is empty),
- security of usage (no local cache or full cache history, respectively),
- camera usage (lots of photos taken with the device versus almost none (i.e. the user might posses a digital camera); photos taken might be uploaded to Dropbox or similar (i.e. information on user's online/backup preferences),
- . . .

5.1.3 Protectable Data

Basically, any data can be protected by simply erasing any trace of it. But, this would be very utopian and impractical. Firstly, data transmitted from the clients to the servers must be somehow received and intermediately stored by the servers. Hence, the actual transmission information is not protectable in the beginning, so during reception and processing, the data cannot

[65] One can determine which client transmits what, how much and when, exactly.
[66] The list is sorted alphabetically and does not claim to be exhaustive.
[67] Personal Area Network
[68] Deskpets International has a connectable remote control device to remotely control little cars, tanks, robots, etc.
[69] PayPal has a connectable NFC-device for in-shop purchases

be protected. However, *after* 'interesting' data has been extracted from the data transmitted, the actual reception as well as the received original data can be deleted and hence protected. Also, some data may be required and should not be protected by deletion. For example, should the service provided be used maliciously, a very high interest in the communication data arises, especially the ones allowing identification of the culprit. Practically, many data protection officers encourage a two step storage philosophy: data (especially communication data) should be stored as parsimonious as possible with reduced access privileges so that only an accountable number of individuals can access the data. Should an infringement occur, this limited number of individuals should access the data and only provide excerpts which are relevant to the infringement[70].

5.2 DATA REQUIRED BY MAPBIQUITOUS

5.2.1 MapBiquitous: Data Collection, Storage, Processing and Protection

As it is not foreseeable which applications users may install on their mobile devices, the review of collected data shall be limited to MapBiquitous aspects. In order to provide sufficient navigation performance, MapBiquitous currently is collecting either directly or indirectly the following data:

- GPS data transmitted to the server (used for the fetching of map or building data),
- accelerometer data (experimental effort to enhance the localisation [Gru12]),
- compass data (experimental effort to enhance the localisation [Gru12]),
- connection data such as IP addresses, TCP stream identifiers, etc. used to connect the MapBiquitous servers (the servers may log these data in access logs),
- usage frequency (the servers may log any access in access logs),
- users' geo-profile (as the servers may log any access in access logs, preferred abode/whereabouts may be derived (time-precisely)).

In course of the crowdsourced improvement of MapBiquitous proposed in this thesis and in Gerd Bombach's assignment paper [Bom12], the following data are proposed to be collected and transmitted to the servers, as well:

- GSM data (especially on the GSM cells the user is in),
- POI data (new data the users may explicitly create),
- correction data (feedback on map data the users may provide), and
- movement data (information on whether users follow suggested route or not).

The so collected data must only be stored and processed in accordance to the boundaries described in the previous section; hence, the collected data should never be released to third parties, except law enforcement. Further, the data stored should be limited to the very minimum allowing the crowdsourcing process to gather/calculate results as well as abiding lawful boundaries. For example, original submissions should be deleted from the storage after the information has been extracted from the data. Additionally, connection data should be deleted immediately, or even better, never be created in the first place, as far as legal control regulations allow it. Finally, any communication containing personal data should be encrypted, impeding their interception.

Comment

Expanding the concept of how MapBiquitous should store data, it should be proposed to encrypt the storage (e.g. the database), impeding an attacker/hackers capability of directly

[70] I.e. not an entire log should be handed over to the police or prosecution, but only log-lines relevant to the infringement case.

accessing the information stored within the data in the time between submission of the data and the time they are deleted. For the current proof-of-concept implementation of MapBiquitous (23 September 2012), only the passwords are actually encrypted; therefore, a corresponding passage is included in the legal disclaimer presented to the user when activating crowdsourcing on their client device.

5.3 CONCLUSION

To the best knowledge of the MapBiquitous team, legal boundaries of German laws must be accounted for by the MapBiquitous. The manner in which data is ascertained, stored and processed must comply with German laws and any user should be made aware of the ascertainment, storage and processing of the data at hand; at least the users are required to confirm a corresponding legal disclaimer when using the MapBiquitous system.

A not exhaustive list of design goals emerges when considering diverse aspects of desired system functionality and legal boundaries:

1. Possibility to not participate in the crowdsourcing.
   (a) The system shall remain usable as to now.
   (b) Are modifications inevitable, they shall be as minimal as possible.
   (c) Only basic data without amendment of crowdsourced corrections/additions shall be accessible.
2. Choice to participate in the crowdsourcing.
   (a) Legal boundaries of the Federal Republic of Germany (BDSG) are to be adhered.
   (b) Aware crowdsourcing (A∗C) with crowd awareness shall be supported.
   (c) Unaware crowdsourcing (U∗C) shall not be obstructed.
   (d) Participation shall be anonymous against the crowdfunding building servers.
   (e) Submissions shall be delete/remove/erase/revoke-able ('*drer*able').
   (f) Submissions shall only be drerable for a defined time.
   (g) Submissions shall be reviewable.
   (h) Submissions shall be siftable.
   (i) Submissions shall lead to direct feedback.
3. The crowdsourced system architecture must be extensible and secure.
   (a) Required application logic shall be easily replaceable, e.g. by loading a module.
   (b) The source code of new components shall be self explanatory and easily extensible rather than highly optimised.
   (c) Data ascertainment must be extensible and configurable.
   (d) Communication must be encrypted.
   (e) Data stored on servers must be encrypted.
   (f) User shall be bound to infrastructure components only loosely.
   (g) GSM data shall be ascertained and stored for future processing and information extraction.
   (h) As proof of system function, at least one crowdsourcing method (position correction) must be functional at the end of the implementation time.

Obviously, goal 1a excludes goal 1b, but not vice versa. Further, the goals 2e and 2f are inextricably bound reciprocative. Goal 2g has potential to obstruct goal 2i; hence, a clever solution obstructing neither of the goals too much must be found. Lastly, goal 2h is strongly correlated with goal 2e, as submissions that are considered 'sifted' should not be drerable. A more comprehensive overview of the relations between the first two sets of goals can be seen in Figure 5.3.1. The third set of goals is not considered in the figure as it surmounts the other two sets. This especially means that goals originating in the first two sets may be softened or even abandoned should they endanger on of the goals from the third set.

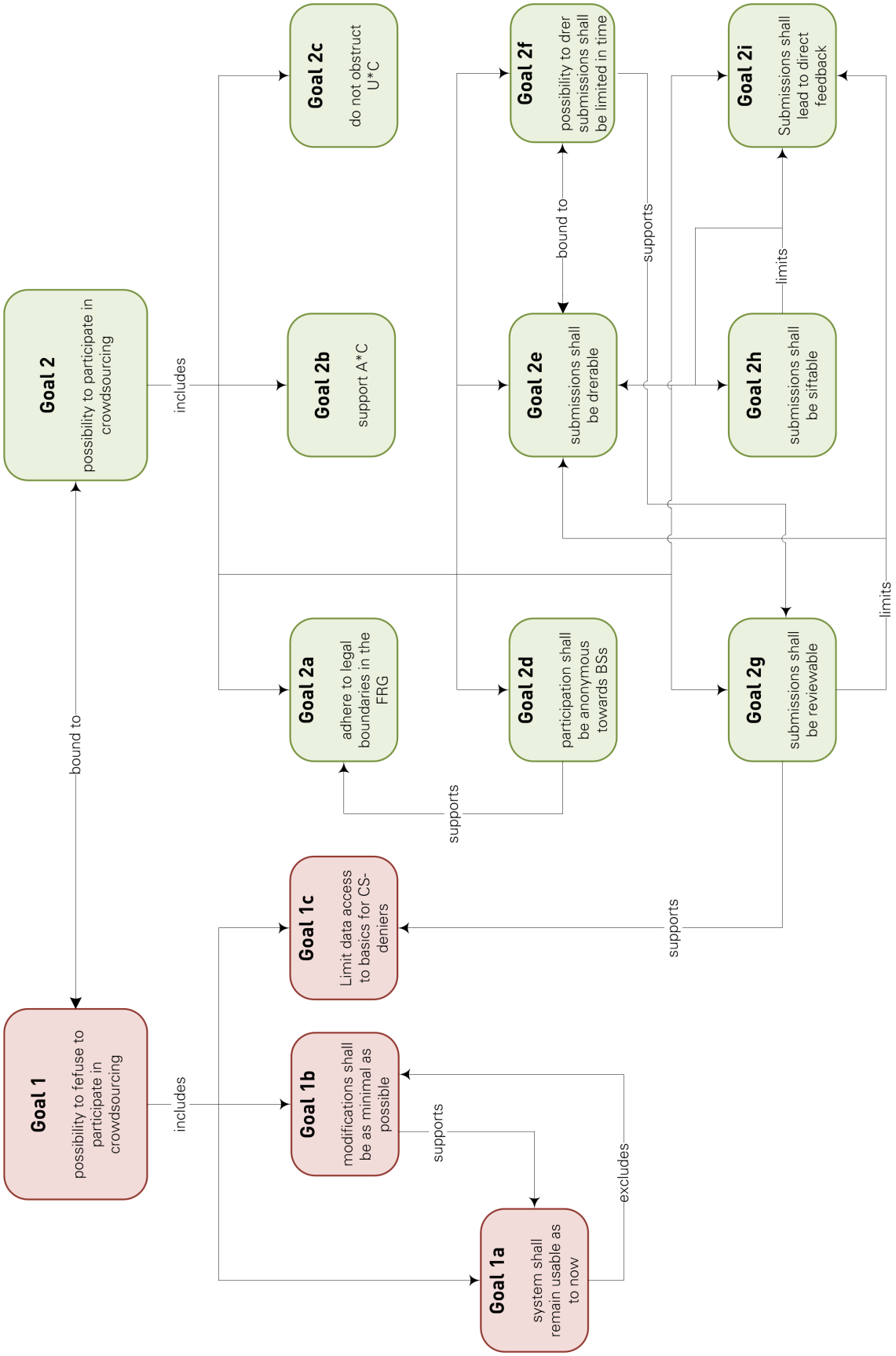


Figure 5.3.1: Overview of the relations between the different design goals

Goal 2d is not limited to anonymous access, but should be considered in a broader context as the possibility to access a building server without exposing one's identity. Of course, the building server should be able to recognise actions originating in the same user, ensuring goals 2e as well as 2g, and especially goal 2i.

The so defined list of design goals is considered not to be exhaustive, as further requirements may arise while designing the conceptual extension of the MapBiquitous system as well as when actually implementing the concept. Therefore, requirements which may arise after the herein given requirements review is concluded shall be recorded at the point they arise[71].

[71] Requirements that are obviously important shall be clearly marked as 'new requirement', but any other requirement shall be implicitly given by the usage of subjunctive/conjunctive phrases.

6 CROWDSOURCED OPTIMISATION CONCEPT

「理由は百年にブリッジをするためにどのような取り到するための一夜の信仰の翼を取るだけ。」

神のメモ – アリス

CROWDSOURCED OPTIMISATION CONCEPT

Currently, MapBiquitous lacks a reliant source of map data and/or corrections of the same. Of course, the basic concept of having building owners provide ichnographies, floor plans, POIs, etc. of 'their' buildings is still valid, but the time until updates and/or corrections are applied into the map on the directory server and/or building servers may exceed acceptable limits, hence incorrect data may supersede practical data. A radical new approach would be to include the MapBiquitous users into optimisation process. A very straightforward approach would be to make use of crowdsourcing, turning all users into the crowd, whereas the directory server and the building servers would act as crowdfunders. The actual data to be collected by the crowd would be user generated quality feedback and corrections (explicit crowdsourcing) on the one hand, and derived information from user observation data on crowdfunder-side (implicit crowdsourcing) on the other hand. The conceptual design idea of a crowdsourced MapBiquitous is depicted in Figure 6.0.2.

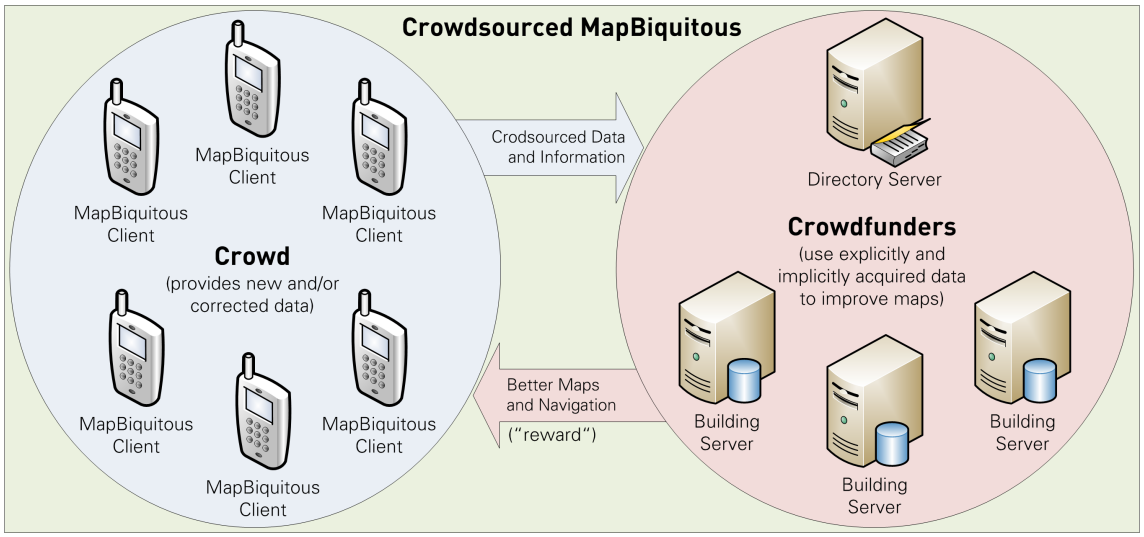


Figure 6.0.2: Conceptual design of a crowdsourced MapBiquitous

Looking back at the requirements identified earlier in chapter 5, the division into crowdsourcing clients and crowdfunding servers does not affect design goal 1, as users denying participation in the crowdsourcing could be considered crowdsourcing clients that do not hand in any submissions (passive participant). In this sense, the actual crowdsourcing interface between crowd and crowdfunder should only be added to the existing system, ensuring goals 1a and 1c are obtained. In the best case, goal 1b will never be of consideration as goal 1a excludes goal 1b, or the other way round, goal 1b makes goal 1a unobtainable.

Goal 2d forces the usage of a proxy handling requests between the clients and the building servers and directory service respectively, since any direct communication between them would expose the client's identity[72] to the servers. For obvious reasons, the goals 2e, 2g, 2h and 2i can only be obtained by usage of the proxy as well, since any submission needs to be proxied due to goal 2d. Therefore, only the proxy is able to link submissions to their submitters; hence, drering a submission or all submissions of a certain user is only possible with help of the proxy, and only the proxy is able to gather intermediate result from the crowdfunders in order to give direct feedback to submissions. Obtaining goal 2f in this context is actually easy: should the proxy request drering a submission, the crowdfunder simply denies. Of course, the proxy should inform the client of the denied drering.

[72] At least the current IP-address and all information derivable from the IP-address.

As goal 2d was added by the authors, it is fair to consider modifications to the existing system under the premise that goal 2d does not actually exist. Even then, the goals 2e, 2g, 2h and 2i clearly lead to the demand of a proxy handling the submissions. Should there be no proxy, all crowdsourcing data must be stored either on the clients, the building servers or the directory server. Firstly, storing all data on the clients is unfeasible for a variety of reasons, including but not limited to accessibility problems, such as a crowdfunder wanting to review all submissions made for a certain point of interest; what to do while clients are offline and hence their data inaccessible? Secondly, storing the building server data on the corresponding building servers is unfeasible for a wide range of reasons, too. For example, building servers would need to communicate with each other in order to collect all submissions made by a user, automatically turning the directory service into a communication bottle neck as all building servers cannot know each other; hence, having have to look each other up via the directory service. Further example, the availability of submission data is at risk should a building server be unreachable or abandon service. Then, storing submission data on the directory server may be considered feasible in terms of centralised accessibility, but it is unfeasible when considering the accrued communications bottle neck. Also, it could proof infeasible when accessing the directory server from different geographic regions. Anyhow, having have to modify the directory server vastly, maintaining goal 1b would be utopic.

Having the building servers actually process the information contained in the submissions and leaving any decision concerning the information to the building servers ensures compatibility with both goals 2b and 2c. It is most important to actually regularly make the users of the clients aware of the aware crowdsourcing in order to properly obtain goal 2b, e.g. by regularly displaying a corresponding disclaimer, and ensuring that consent given by the users has limited time of validity.

Lastly, anything to be added to the system shall be designed from start to fulfil goal 2a by limiting transmitted and stored data to the very minimum possible.

All of the concepts and ideas that are going to be introduced in this chapter can be found in the summarising Figure 6.0.3 and Figure 6.0.4, wherein necessary modifications to the existing MapBiquitous architecture as well as proposed communication paths are highlighted. As a reminder, the original MapBiquitous architecture can be found for comparison in section 1.3 in Figure 1.3.1.

6.1 CROWDSOURCING CLIENT

As mentioned earlier, the conceptual idea for the crowdsourced MapBiquitous divides the system into crowdsourcing clients and crowdfunding servers. As the latter is extensively discussed within this thesis (refer to section 6.2), the first is extensively discussed in Gerd Bombach's assignment paper ([Bom12]) and shall not be presented within this thesis. The interested reader if kindly refered to [Bom12].

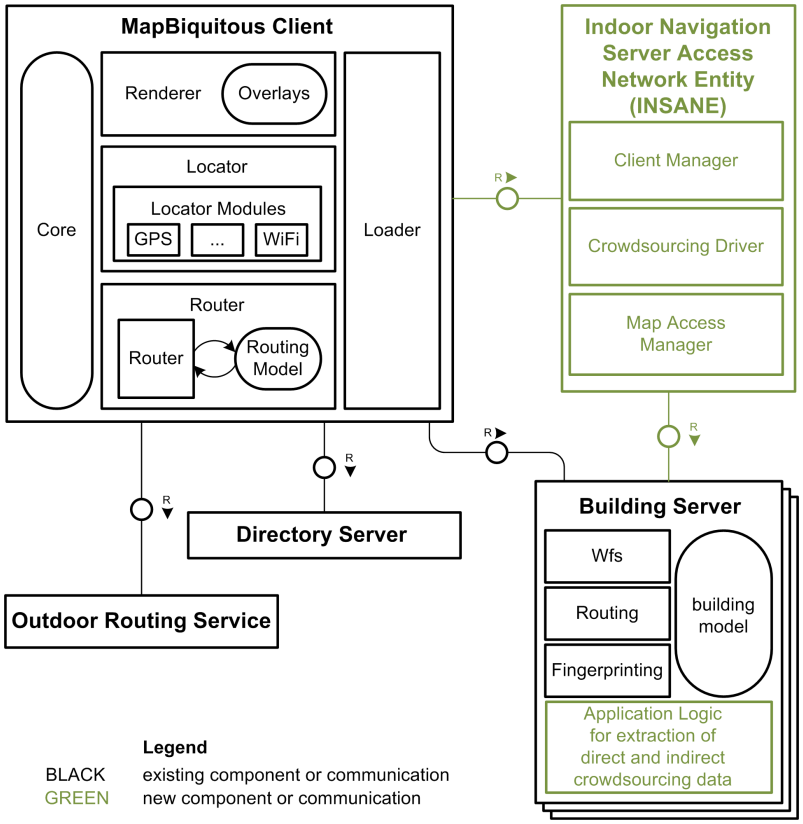


Figure 6.0.3: Simplified concept of proposed modifications to MapBiquitous' architecture in order to implement crowdsourcing

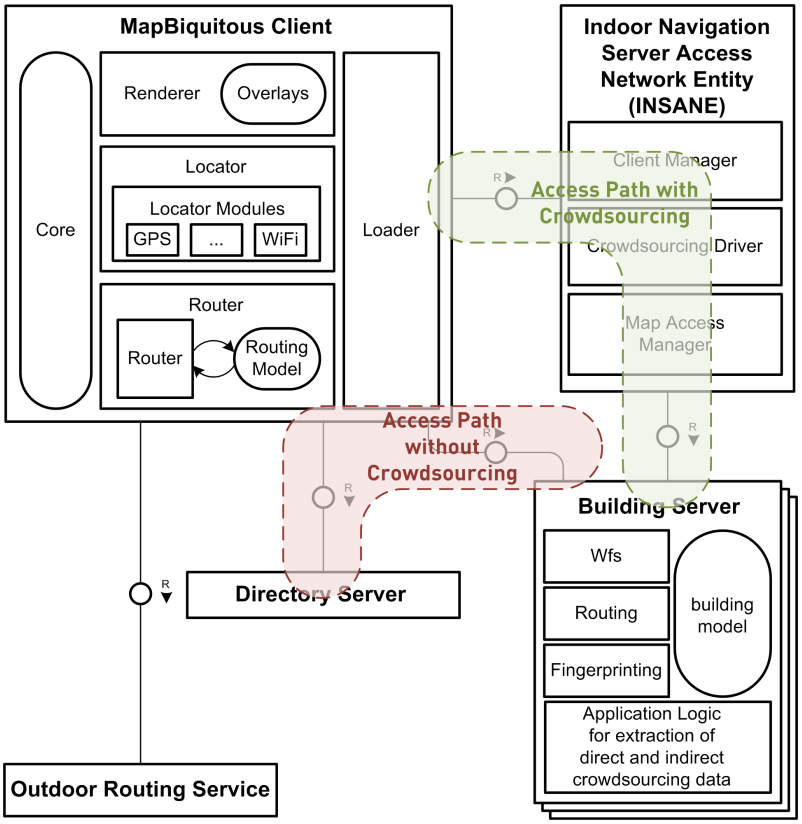


Figure 6.0.4: Display of the communication paths in the simplified concept of proposed modifications to MapBiquitous' architecture – Note: from the client's view point, the directory server is no longer required when following the crowdsourcing access path; of course, the INSANE may or may not request directory information from the directory server, but this is not part of the access path

6.2 CROWDFUNDING SERVER

Independent of the data collected at the crowdsourcing client, a reliable server-side crowdfunding must take place. Reliability in this context is not limited to 'reachability', but also includes legally binding and reproducible storage of the data at hand (refer to section 5.1). The basic idea now is to store as many information as possible with as few data as necessary. But, the problem is what can be considered necessary data. As this question cannot be answered satisfactorily as it is not known which information is deemed 'wanted', all data collectible should (partially) be storable on the crowdfunding server, ensuring extensibility of the crowdsourced information retrieval at a later time. Of course, this rises data protection issues, so it shall be suggested to store all data anonymised, but still related to the clients, making multiple submissions of the same crowdsourcing data from the same client impossible. Further, any submissions shall be signed using asymmetric crypto functions, so that submissions can be proven towards the clients and servers.

Comment

> *For this section, it shall be of imperative understanding that the crowdsourcing as designed within this thesis only affects the indoor navigation capabilities of MapBiquitous. The outdoor component shall remain untouched (refer to figures 6.0.3, 6.2.1 and 6.2.19).*

The remainder of this section shall extend the basic modification concept introduced earlier (refer to Figure 6.0.3) by contributing to identification of the crowdsourcing clients while maintaining their anonymity towards the building server as well as protecting the communication from intrusion/interception with help of encryption. Then, modifications to the building servers will be discussed before introducing a new entity to the system, the INSANE. Access control issues on side of the building server as well as required database structures are thereafter discussed before concluding with suggestions to optimize the beforehand introduced INSANE. – All extension and modification[73] that are going to be introduced are displayed in Figure 6.2.1.

6.2.1 Anonymity and Identification of Clients

The easiest way to accomplish the design principle of anonymity towards the building servers – as a reminder: goal 2d – is to not transmit any personal data of the clients at all; hence, only using a pseudonymised means of identification when communication should take place. This pseudonymisation shall be easily reproducible, delivering always the same identification tokens, but irreversible, so that the building servers cannot obtain information beyond the scope of the original submitter identity, such as their IP-address, their registered email-address or other personal data that may be stored within the crowdsourcing participants' user-profiles. Another aspect of anonymisation, or at least not gathering unrequired data, is not to transmit the clients' hardware identifications into the system. The easiest way to guarantee this is by not transmitting any hardware information at all, especially when considering the problems described in [Bec12]. Unfortunately, different devices may contribute different measurands to the crowdsourcing[74]; hence, it may be imperative for the crowdfunder to be able to differentiate between a user's different devices. When considering the disclosure of the existence of different devices of a user, hash functions seem to satisfy the anonymity desire toward the unique hardware identifiers; therefore, identification tokens to separate one and the same user's different

[73] The architecture in Figure 1.3.1 displays the fingerprinting service as part of the building servers; however, the actual implementation by S. Gruna [Gru12] uses a dedicated fingerprinting service separate from all building servers. This is accounted for in Figure 6.2.1.

[74] E.g. devices of different manufacturers provide different measurands for the same location when calculating the corresponding WLAN fingerprint.

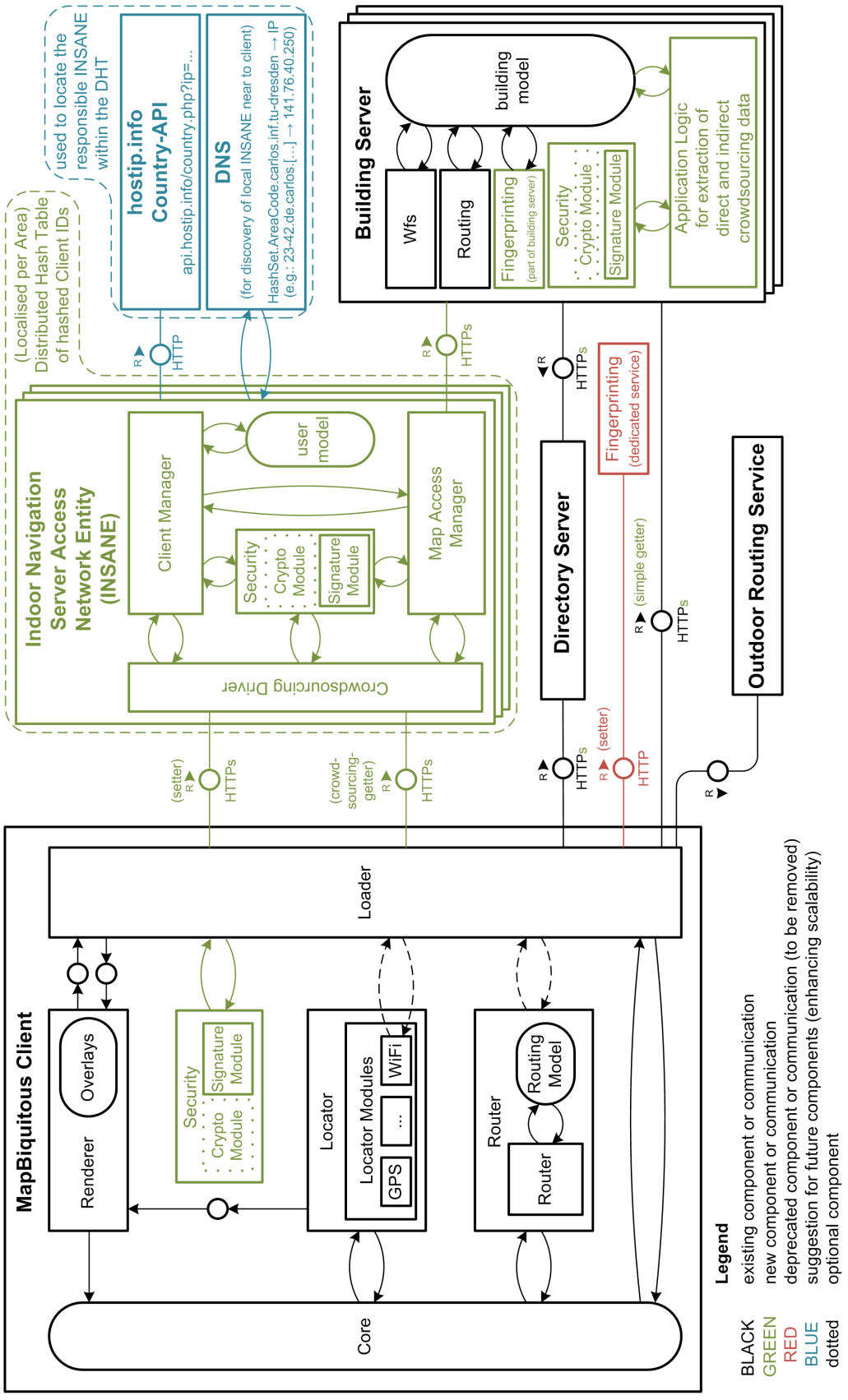


Figure 6.2.1: Concept of proposed modifications to MapBiquitous' architecture in order to implement crowdsourcing

devices shall be calculated as a SHA-256 value[75] of the crowdsourcing client's unique hardware identification, which should automatically bind the token to the hardware of the client, allowing to maintain the same token even if the SIM or the operating system of the client should be exchanged, while each individual client device of a user has a distinct identification token. However, this concealment of the hardware-ID should be done in parallel to the use of submitter-pseudonymisation; therefore, the hardware-ID is concealed, but still linkable to the individual user. Considering goal 2i paired with a possible future demand of extracting information from the crowdsourced data that is ascertained only by certain device types, this is totally okay and supportable.

Comment

> *As a reminder, the system should work with only a username and a password, but the proposed device/client identification token may be of use should a differentiation of different devices of the same user be required. For the remainder of this thesis, it shall be assumed that such a differentiation is required.*

The usage of SHA-256 instead of MD-5[76] shall ensure absence of collisions[77] as well as simplicity of calculation. SHA-256 allows a maximum input length of $2^{64} - 1$ bits (round about 2 Exbibyte), always outputting a string of 256 bit length, while computation performance is only reduced to about 40% of MD-5 performance[78]. Of course, usage of SHA-512 would enlarge the set of possible hash values, but the calculation principle is the same as for SHA-256. On 3 October 2012 the United States National Institute of Standards and Technology (NIST) proclaimed the successor of SHA-2 – the hash family which SHA-256 belongs to – by presenting SHA-3[79] which is based on a totally different algorithmic approach. Therefore, the usage of hashes is conceptually forced, but SHA-256 shall only provide as an example.

The above mentioned hardware identification token should not be limited to the International Mobile Equipment Identity (*IMEI*) which all GSM-devices posses, as actually only GSM-devices can provide it. Instead, whatever unique hardware identification provided by the device itself should be used. For devices operating under the Android operating system, this hardware identification depends on the primarily used connection type; especially, the IMEI is provided when GSM is present. Additionally, UMTS-based identification, etc. is provided.

Further, identity theft should be prevented by using passwords in combination with a username and the identification token of the used client. This will be important when expanding the Map-Biquitous project to interact with or be part of other projects that include social community aspects, such as the usage of names, avatars, mail addresses, etc. As password should be chosen by the human users themselves, it is natural to suggest an appropriate implementation of a user-choice password-generation.

Comment

> *For the proof of concept, the IMEI is a simple means to calculate an unique client identification token while maintaining anonymity, but for in vivo implementations it is not feasible, as GSM communication is not encrypted; hence, everybody intercepting the GSM radio waves would be able to calculate the client identification token.*

As the original idea of Gerd Bombach and the author of this thesis stipulated the usage of the IMEI, the rough idea shall be noted before presenting the final draft. – The IMEI is an (hopefully) unique number, allowing identification of mobile phones in GSM-networks, which is normally branded into the devices hardware. As such feature, modification of the IMEI is liable to prosecution in some countries. All devices manufactured as of 2004 (which should include all smartphones) build their IMEI from

[75] SHA-256 is one of the algorithms suggested in the SHA-2 set of cryptographic hash functions.
[76] Message-Digest Algorithm 5
[77] Collisions have been proven for MD-5. Therefore, experts advise not to use MD-5 any longer.
[78] According to 'Crypto++ Benchmarks'.

- 8 digits – Type Allocation Number (TAN) – consisting of
  - 2 digits Reporting Body Identifier (RBI) – the technical accreditation body,
  - 6 digits Approval Licence (ALN) – the accreditation identification number,
- 6 digits – Serial Number (SNR) of the device, as well as
- 1 digit – Check Digit (CD) – not to be transmitted into GSM-network; replace with 0.

The idea would have been to calculate the client identification token as SHA-256 hash of the IMEI and the client password as SHA-256 hash of the SNR. Unfortunately, as mentioned above, the IMEI is not always available. Especially – as the proof of concept implementation is planned to be conducted for clients operating under the Android operating system – the hardware identification provided by the `android.telephony.TelephonyManager.getDeviceId()`-method[80] delivers disjunct values, such as the IMEI for GSM devices, MEID[81] for CDMA devices, etc. The format should be similar, often following the IMEI schema '`########-######-#`', but disjunct variants such as '`##-######-######-#-##`', etc. are also possible. Therefore, the modified concept envisages stripping all dashes ('-'), slashes ('/'), etc. from the hardware identification and calculating the SHA-256 hash of whatever remains, presumably digits. The result shall then be the client identification token to be used when communicating with other entities in context of MapBiquitous. Further, the substring starting at position 6 of the stripped hardware identification shall be used to calculate the client's default password by hashing over it in the same way as done for the client identification token.

Comment

The usage of the IMEI (or on some devices: UDID) must be very carefully monitored, as sloppy usage yields a security risk, as shown e.g. in [Kir12].

In summary, the following must be calculated/generated at the crowdsourcing client's side:

Definition 6.2.2 – User and client

A human or other (intelligent) entity utilising the services provided by the MapBiquitous system is considered a user. Any (uniquely identifyable) device utilised by the user as interface device shall be considered a client.

Definition 6.2.3 – User-individual username

The user-individual username of a user is an arbitrary string over a predefined alphabet of predefined maximum length, iff the selected string allows bijective mapping of the user's identity to a corresponding database record.

Definition 6.2.4 – Client-individual identification token (clientID)

The client-individual default identification token is a string over the hexadecimal-alphabet {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *a, b, c, d, e, f*} calculated by the client. It is the SHA-256 hash of the client hardware identification stripped of its dashes. In Pseudo-Java, the calculation follows:

```
deviceID = android.telephony.TelephonyManager.getDeviceId();
clientID = sha256( deviceID.replace("-","") );
```

[80] `http://developer.android.com/reference/android/telephony/TelephonyManager.html#getDeviceId%28%29` – Accessed 27 August 2012

[81] Mobile Equipment Identifier

Definition 6.2.5 – User-individual password

The user-individual password of a user is an arbitrary string over a predefined alphabet of predefined maximum length. The selected string is not required to allow bijective mapping from client identification to password, but it must be surjective.

Definition 6.2.6 – User-individual submitter identification token

The user-individual submitter identification token of a user is an arbitrary string over a predefined alphabet of predefined maximum length. The selected string is required to not allow mapping to the username or password.

Exemplarily, the following could be considered: A user chooses to use the username '`mustermann`' and the password '`password`' and the device they use has the IMEI '12345678-123456-8'. The given information could lead to

- the user-ID '`e32a370b7912ad78cc6a88fda605a5b3657e9c3b164cee669364aaf3f8cdbb36`', if the user-ID was created by calculating the SHA-256 hash of the username,
- the client-ID '`a6726e1f5cbea0bcf53ee9feeb05e5f80d2a42cb16c0892d94b44c2ed99628-d5`', if the client-ID was created by calculating the SHA-256 hash of the reduced IMEI (as described above), and
- the submitter-ID '`c231a712af7413a7f5e22e35f376a17aaf8b44e0aaf46352bef4086c918-9eb19`', if the submitter-ID was created by calculating the SHA-256 hash of the UNIX-timestamp of the current time (e.g. 1349293093) concatenated with the username concatenated with a random integer (e.g. 17742).

6.2.2 Safety from Interception through Encryption

However, the entire calculation/generation of hashes and selection of usernames and password is meritless, should the hashes be transmitted in plaintext between client and server. Therefore, encryption of any transmitted data should be considered. There are two ways to reach the design goal of encryption; firstly by implementing a crypto-suite, and secondly by relying on ready to use solutions.

The first idea can be ruled out quickly, as the generation of large prime numbers required by the Diffie-Hellman algorithm to exchange symmetric keys over an unsecure channel is not feasible on mobile devices such as smartphones, especially Android devices. Android uses a partial *Bouncy Castle* package to handle most of its cryptography, but this Bounce Castle implementation is by far not complete. Therefore, programmers often prefer to use other cryptographic packages such as *Spongy Castle*, but those packages, i.e. non-Android Java crypto libraries, are often not entirely supported by Android as Android does not support all Java packages those packages might require.

Should all required packages be available and supported, feasibility may be given as following short example[82] proofs: Overall, using Diffie-Hellman a shared key for encryption of communication shall be exchanged, which's length shall be limited to 4096 Bit, or 512 Byte. At first, 'large' prime numbers must be generated in order to 'secure' the actual Diffie-Hellman-based communication. The question what 'large' or 'large enough' is in context of prime numbers shall be left unanswered here; rather, two prime numbers[83] of length 12 shall be chosen and deemed 'large

[82] The example should be easily derivable as high expectations towards computation power for encryption should be generally known; nevertheless, this example is based on the information available at `http://java.sun.com/developer/technicalArticles/Security/AES/AES_v1.html` in combination with `http://www.helloandroid.com/tutorials/encrypting-your-data` and `http://www.androidbenchmark.net/cpumark_chart.html` – Both accessed 27 August 2012

[83] For example, 123456789011 and 987654321097 could be chosen.

enough'. Random generation of each of the numbers with subsequent prime number test[84] on a HTC Desire Z takes an average of 40µs when computed exclusively, which is actually fast. Even when not being computed exclusively – as the Android devices are actually meant to be phones and also numerous background programmes must be computed multi-tasked – only 80 to 100µs pass. The server may compute its prime number even faster. The following encryption using the exchanged key can be conducted within a few µs, as well.

Further, any communication to take place has to be encrypted and/or decrypted by an dedicated module, while signatures must be generated/verified, as well. Basically, an implementation of a key exchange similar to Diffie-Hellman could be considered, e.g. using general random numbers instead of primes, etc.

The second idea can be very easily implemented: As MapBqiuitous already uses HTTP-based communication between client and server, the easiest way to acquire encryption is to use HTTPs instead of HTTP. As most mobile device operating-systems, such as Android, support HTTPs natively, only a few modifications to the source code on client-side are necessary. On server side, normally some sort of server service is running, e.g. Apache Tomcat. Commonly, these service implementations support HTTPs natively, so no modification of server side source code is required in order to implement HTTPs-based secure communication. Hence, the only actual problem to be solved on both sides is the implementation of methods for reproducible and verifiable signatures. The performance of the certificate-based HTTPs encryption[85] is even more feasible than – if implementable – Diffie-Hellman on Android.

Now, after having found basic concepts for anonymisation and encryption of the communication, the actual storage on server-side shall be focussed in the following subsections.

6.2.3 Unaltered Directory Server and Modified Building Server

In order to maintain scalability of the system, it is of no advantage to store crowdsourcing data on the building servers or the directory server (refer to Figure 1.3.1 for details on building servers and the directory server). – Firstly, the directory server as discovery component for responsible building servers has nothing to do with the crowdsourcing process; therefore, no data needs to be 'written' to the directory server by the clients. Hence, focus on 'write-access' can only be applied to the building servers, or in more implementable words: Besides a simple setter used to add new building servers, the directory server only requires getter methods. As these are existing already, no modifications to the directory server are required. Considering the demand for encryption to be introduced in the previous subsection, the communication protocol for communication between clients, the directory server, and the building servers should be changed from Hypertext Transport Protocol (HTTP) to Secure Hypertext Transport Protocol (HTTPs). But, luckily this only requires setup of an SSL certificate and a one-line modification to the server configuration.

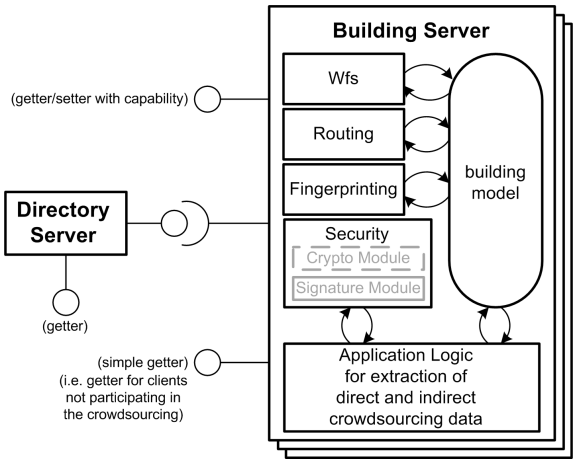


Figure 6.2.7: Modifications to the building server

[84] I.e. to check whether n is a prime number or not, each number from 2 through $\sqrt{n}$ must be analysed whether it is a factor of n. For a 12 digit number this takes a maximum of $2^{12} = 4096$ distinct factor tests.

[85] Transport Layer Security (or antediluvian Secure Sockets Layer) uses certificates and random numbers which are not prime.

Secondly, as the herein discussed[86] 'write-access' only affects the collection and storing of information gathered by crowdsourcing – e.g. in order to correct points of interest – any 'read-only access' to the building servers should maintain untouched. This leads to an asymmetric access characteristic, with untouched getter methods and newly created setter methods. Hence, the building servers must be modified in order to accommodate write-access. Even further, a security concept is necessary, defining who is allowed to write-access which data, so access control lists (ACL) and/or capability-management should be introduced. As a consensus of scalability and ease of modification of the existing components, ACL should be implemented on building servers, while reduced capabilities should be maintained by another entity either central or decentral. These reduced capabilities have nothing in common with what generally is defined as capability except for the fact that they allow clients to easily grasp which access rights they have on which server.

As is true for the directory server, communication should be encrypted; additionally, any write-access needs to be logged and made legally binding[87] and reproducible[88], so signatures should be implemented alongside encryption. The three design-goals mentioned – ACL, encryption and signatures – can be handled within one new component to be implemented, the security module.

Additionally, the in any event existing ACL can also be used to grant certain reader privileges; hence, adding different levels of accessible data via getter methods. E.g. the display of sensitive infrastructure information, such as the location of server rooms, can be limited to users with corresponding rights.

The setter methods on building servers are not supposed to write crowdsourced data directly to the building servers as this has a negative effect on scalability, as all data is spread over the set of all building servers, making it (almost) impossible to gather the data from all building servers, or a selected subset of them, in a glimpse... The introduction of an additional entity can provide the desired scalability and shall be introduced in the next subsection.

Finally, after having data communicated to and stored on the building server, the data must somehow be utilised. For that, an application logic is required which can work on the direct crowdsourcing data stored and extract further information from the indirect crowdsourcing data stored. This cannot be done either on the client nor INSANE (introduced in the next subsection), as computations with and on the data collected and stored must be conducted in context of modifiable original data, which is only the case on the corresponding building server itself. Further, the application logic implements the task of deciding what data to make use of and how, as well as what data not.

[86] Of course, there also exist a setters for the creation of POIs which work independent of the crowdsourcing process, namely using the geoserver's transactional web feature service.

[87] Any submission must be undeniable. Should a client deny responsibility for a submission, the server should be able to proof that the submission actually originated at the client in question.

[88] Any submission must be verifiable. Should a server acknowledge a submission but appeal content or timing of the submission, the client should be able to proof content as well as timing.

6.2.4 Indoor Navigation Server Access Network Entity

As mentioned earlier, the introduction of a new intermediatory component, the 'Indoor Navigation Server Access Network Entity' (*INSANE*), may provide the desired scalability by ensuring that any write-access to the building servers is intercepted and centrally prepared for storing and processing on the building servers. Additionally, a backup/redundant storage of the submissions on the INSANE is to be considered in order to allow scalable search for submissions on a user or client device basis, without sending requests to the building servers. For this purposes, the INSANE shall consist of a crowdsourcing driver encapsulating any crowdsourced read- and write-access oriented towards the server-side. In cooperation with a security module for encryption and signature-handling, the crowdsourcing driver would forward client related access to a client manager, and building server related access to a map access manager. The latter should conduct the actual call of setter methods on the building server, while the client manager would conduct any access to stored client data, such as the user model representing the accessing client. As such accesses should be legally binding and reproducible as defined for the client access itself, it is necessary to ensure that all involved components make use of appropriate security modules, to the very least ensuring that all[89] communication is signed and encrypted.

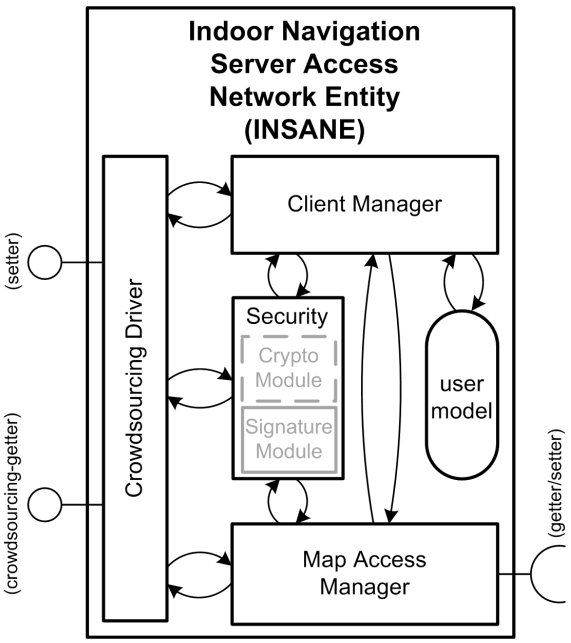


Figure 6.2.8: The INSANE – a new component

Maintaining anonymity of crowdsourcing submissions towards the building servers should be of imperative nature, as well. Therefore, any write-access to the building servers should not only be proxied via the INSANE, but also resigned and re-encrypted, ensuring that any submitted data is traceable from the building servers to the INSANE, only. Nevertheless, the INSANE should store any resigning and/or re-encryption, maintaining a legally binding reproducible trace of the data submitted. Anyhow, access to the resigning and/or re-encryption data should be limited to authorised entities, such as the police or crown prosecution service.

Summarising, the INSANE acts as a proxy for crowdsourcing read- and write-access between clients and building servers, additionally storing all submissions of the clients redundantly. Obviously, the original building data are stored on the building servers; however, any submission is bound to a set of additional information, etc. the identity of the client submitting, etc. These additional data must never be stored on the building servers. Further, the INSANE shall be extensible for future community-related additions, such as a bridge to social communities, offering an extensible database for additional storage of pseudonyms, email addresses, avatar images, etc.

For obvious reasons, granted access rights on building servers should be stored associated with user profiles. Therefore, the INSANE should not only store client related information, but also lists of granted access rights in order to enable any client to gather information on the granted rights centrally; hence, maintaining scalability.

As there exist different terms in context of rights management, such as 'ticket', 'capability', etc, an unambiguous term should be used to identify granted rights.

[89] Especially communication between the clients, the INSANE, and the building servers.

Definition 6.2.9 – Privilege Pointer

The copy of an ACL-based privilege granted to a user which is distributed from the ACL maintainer is a *privilege pointer*, pointing out access rights the user has on the issuing server. They are a mere remote copy of the information from the ACL; no rights are granted based on them.

Definition 6.2.10 – Privilege Pointer Collection

A set of privilege pointers issued for the same user is the privilege pointer collection of that user. The empty set equals an empty privilege counter collection.

Compliant to Definition 6.2.6, the final decision whether a user is granted access or not, is made based on the ACL on the building servers. Privilege pointers can be created either by the INSANE or a building server. Several use cases are possible for the issuing of a new privilege pointer; they are described in section G.3.1.

All described use cases show a clear demand for means of direct communication between building servers and the INSANE. Therefore, a corresponding interface shall be envisaged. Further, the INSANE should be designed to act as a general access proxy between all users' clients and all building servers, as such a general proxy would be able to allow provision of different data-sets for crowdsourcing participants and deniers. Basically the idea would be to present crowdsourcing participants with an immediate feedback on their crowdsourcing submissions, such as instant (partial) map updates, etc. Further details on this design aspect are described in section 6.3.

6.2.5 Access Control Lists, Privilege Pointers, Resigning and Re-encryption

As seen in the previous subsections, any write-access towards the building servers is proxied over the INSANE. Hence, a per definitionem distributed collaboration of the INSANE and the building servers is required. This is especially true for the interaction of access control lists, privilege pointers and security measures.

Firstly, the access control lists are maintained for each building server on the building servers themselves. Hence, they are of imperative nature for the local security on the building servers, warranting a last line of defence against malicious access from users/clients. As such, the ACL are to be stored on the building server solely and are never to be divulged. Figure 6.2.11 displays the basic concept:

- Any access to the building server yielding rights diverging from the default access rights requires identification of the accessing user,
- the identification is looked up in the ACL,
- should the identification be found, the rights as stored in the ACL are granted/denied, and
- should the identification not be found, the default rules at the end of the ACL are applied, granting/denying default rights.

The described concept considers 'guest access' to be an access without provision of an identification; hence, default access rights apply. Therefore, the maintainer of the access control lists must know whether guests access is desired or not, and consider corresponding default rules.

Secondly, privilege pointers are being used to give users a scalable means of learning about their respective rights on the building servers without divulging the ACL. Of course, these privilege pointers must be protected against fraud, which can be accomplished by having each building server sign the privilege pointers before publishing them. This way, any client can verify that

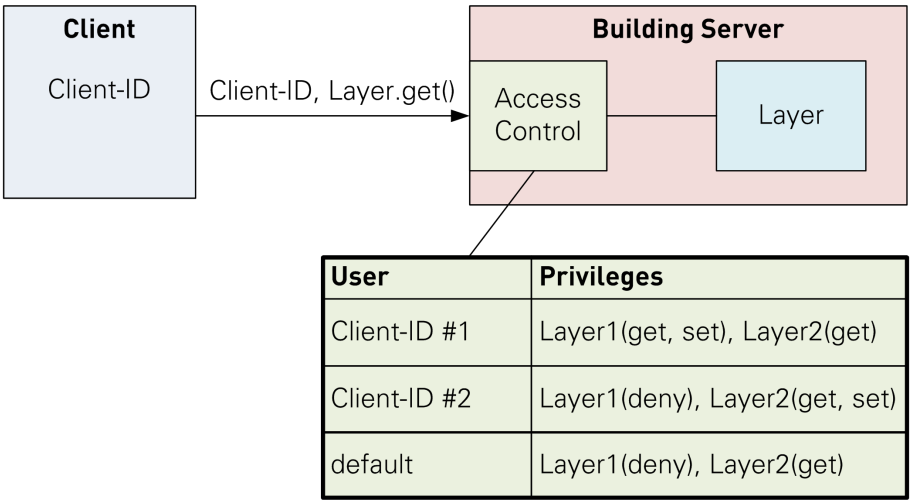


Figure 6.2.11: Access Control Lists are located on the building servers

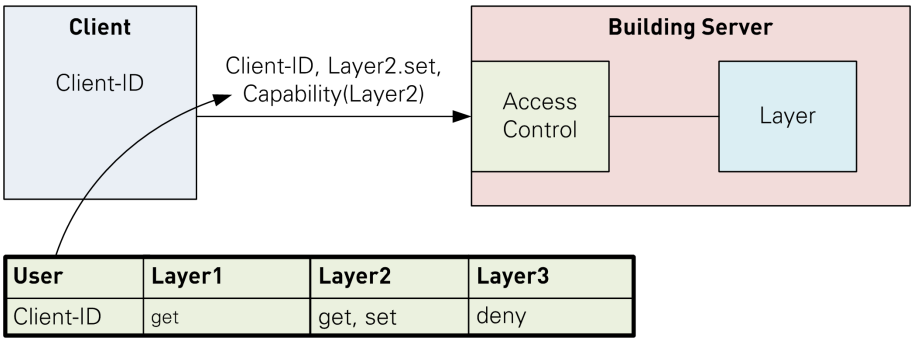


Figure 6.2.12: Capabilities are located on the clients

the information in the privilege pointers are valid. The concept of the privilege pointers follows that of capabilities; their basic concept is displayed in Figure 6.2.12.

Commonly, both types of access control are considered equivalent, as they are based on the interpretation of the resource access matrix as depicted in Figure 6.2.13, but that is untrue as proven in [MYS03]. Instead of actually suggesting to implement the envisaged object capabilities model introduced in [MYS03], a rather simplified combinatory concept shall be proposed:

- building servers shall use ACL-based access control, with any access decision pending a corresponding entry in the ACL,
- clients are provided a capability list for the sole purpose of easily (and scalably) ascertaining on which server their user has elevated (or reduced) rights – this is what was defined as privilege pointer collection in Definition 6.2.7 – and on which not (no privilege pointer concludes to default access rights on the building server with no corresponding privilege pointer present), and
- while accessing a building server, the privilege pointers are cross referenced against the ACL, facilitating recognition of false or outdated privilege pointers.

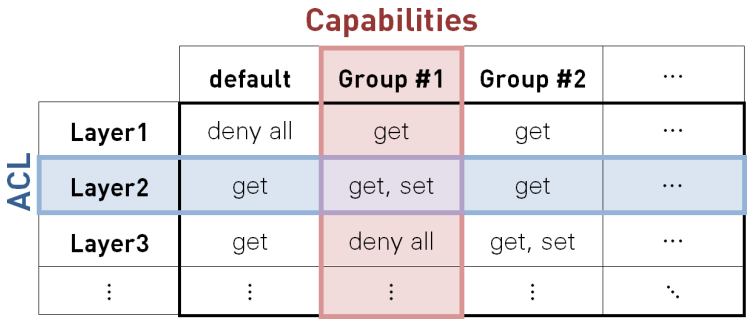


Figure 6.2.13: Interplay of ACL (blue), capabilities (red) and granted rights (purple)

Thirdly, the security measures mentioned earlier must be accounted for. As emphasised several times, anonymity together with reproducibility and security shall be imperative. Hence, anonymisation must be conducted without loss of the latter two, which is only possible by establishing a sustained trust path. The easiest way to achieve this is by using a trust network built on top of signatures and encryption. Of course, signing and encrypting any communication end-to-end would make the entire concept very easy, but that would turn anonymity ad absurdum. Thus, anonymity with trust can only be achieved per aspera ad astra, simplified meaning that the INSANE as a proxy should establish two separate trust paths, one between client and the INSANE, and one between the INSANE and building server. Any further subdivision of the trust path between client and building server shall be inane, as in the end, the client and the building server only trust each other by trusting the INSANE as a proxy; hence, trust toward the INSANE may not be compromised.

Comment

> *Of course, one can argument that the entire concept is keen to be compromised by a man-in-the-middle attack on the INSANE. For this, another implementation detail should be mentioned, without giving the solution within this thesis (refer to 'Future Work' in chapter 9): The use of the Public Key Infrastructure (PKI) for the distribution of public keys is obvious, so the same infrastructure could be used to distribute anonymous public keys of generated shortly valid or even one-time key pairs, only known to the users/clients and building servers. The exchange of the key fingerprints could be conducted over the INSANE using the Diffie-Hellman algorithm (even though this is difficult as stated earlier), excluding the INSANE from end-to-end signature (and possibly encryption, if desired). By this, there would exist two layers of signatures; one over the entire trust path, and one over the separate subpaths.*

The idea of the subdivision of the trust path into to subpaths can be achieved by having the INSANE as the proxy resign and re-encrypt any communication, while removing any identification tokens from the communication at the same time. Simply, when a client desires to write data to a building server, this data is sent to the INSANE as a proxy. The client uses an encrypted channel for this communication and signs their data using their private key. After receiving the data, the INSANE decrypts the data and verifies the signature using the user's public key. If the signature is valid, the data is mapped to a new anonymity token, which allows the INSANE to remap the data to the submitting client and user, should the building server require identification of the user or client for any valid reason[90], but in general the building server will not be able to identify the user or client. The data is then signed using the INSANE's private key and then sent to the building server via a separate encrypted channel. After receiving the data, the building server decrypts the data and verifies their validity by checking the signature using the INSANE's public key. The data are then stored reproducible on the server. The application flow is illustrated in Figure 6.2.14.

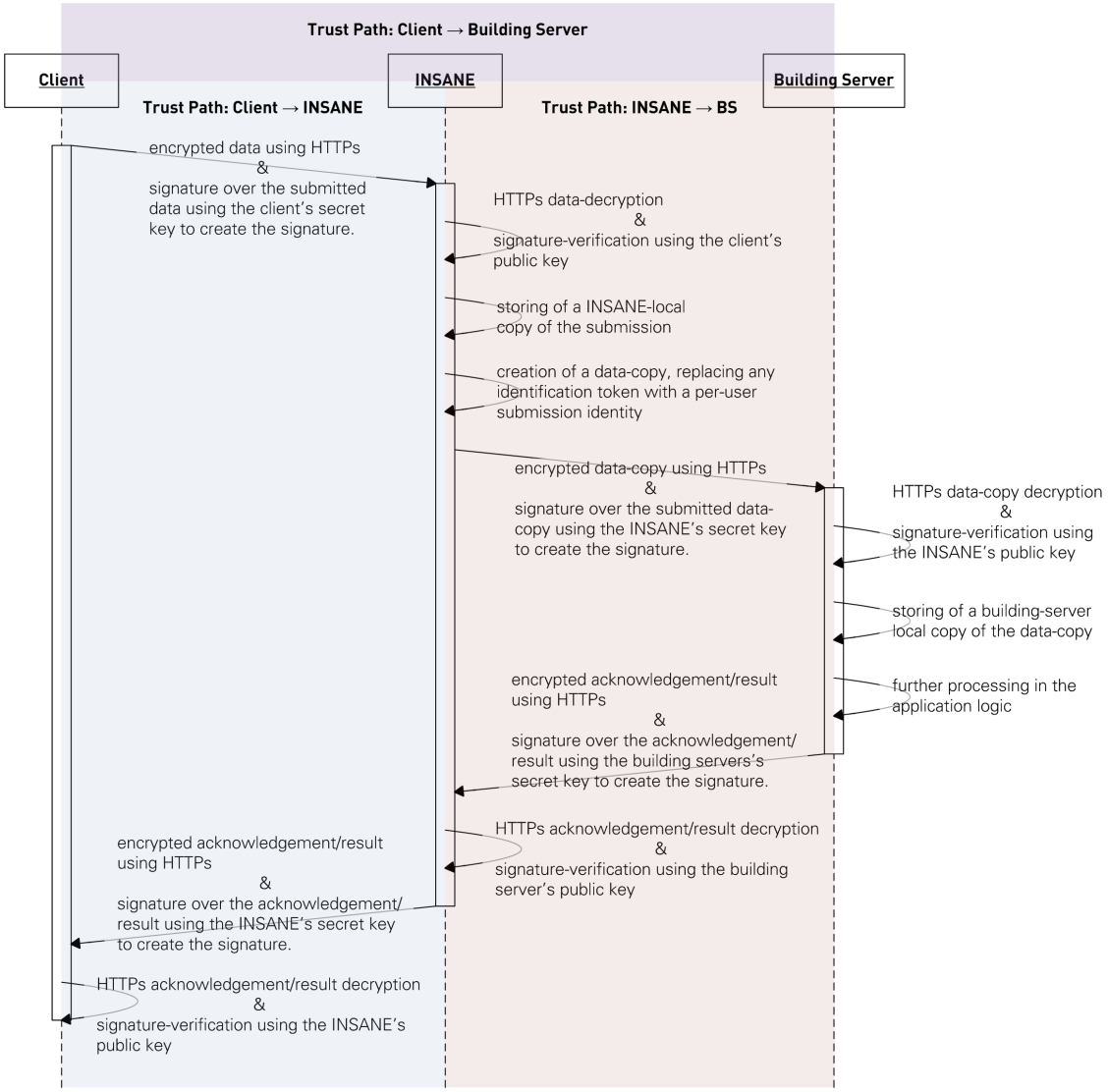


Figure 6.2.14: Conceptual application flow utilising trust paths

[90] For example, the police could be investigating the building server. Should there be a valid legal claim, it is possible to then identify the contributing user and client by cooperating with the INSANE.

6.2.6 Database Structure: Building Server amendment and new INSANE

For the combination of all three aspects into a distributed database structure, the structure depicted in Figure 6.2.15 shall be proposed.

Before moving the focus on the database structure on the INSANE, the modifications and additions to the building server database structure shall be focussed.

Of course, any type of information on the building server may be altered or amended, but for the conceptual introduction of the database structure only modifications/additions (i.e. submissions) to fingerprinting shall be exemplarily. For now, fingerprinting shall be supported for GSM and WLAN. Additionally, corrections of the positions delivered by the WLAN fingerprinting shall be supported. As all submissions are not supposed to modify the data on the building server directly, they need to be stored intermediately until they are sifted. For this to happen, the submissions need to be stored in tables separate from the original data[91]. Also, access rights, known submitters and known INSANEs need to be stored.

Of this set of new tables, the 'AccessRights' table shall be discussed first. This table maps access rights to the ACL of the server. As key the table should firstly present an entry identification, as the table may contain several rows for one and the same access group, since access rights may be invalidated, but remain in the table, while new rights are granted and stored in the same table. Secondly, the actual ACL entry should be contained in or linked from the table. Further, validity of the access rights is stored in a from-till manner. As additional column, a descriptive column shall be reserved for comments, etc.

Second addition to the building server database structure shall be the table containing the allocation of users to submitter accounts. As a reminder, one should note that a user's submitter identification token is used regardless of the device the user uses. Hence, the submitter-ID allows unique identification of the users without breaking the anonymity principle built earlier in this thesis. To ensure that each user is associated with corresponding access rights, or – should there be no association – with default access rights, the table contains the access right identification besides the submitter-ID. Additionally, for quick blocking of users without having have to modify the access control list(s) of the server, a boolean column for blocking information is considered useful[92].

Thirdly, a table designed to store information on INSANEs is added. Beside the unique INSANE identification it contains all necessary information for HTTP communication, namely the host, port and path to the INSANE. Additionally, an information whether the INSANE supports SSL encryption or not is present[93]. Aiming at a faster lookup of the responsible INSANE (refer to subsection 6.2.7), the geographical region the INSANE claims to be in is stored. Lastly, the public key of the INSANE is stored, enabling authentification and encryption.

Lastly, tables containing the actual submissions are designed to hold back the submissions until they are sifted. The main table ('Submissions') identifies all submissions by the submission-ID used by the INSANE handing in the submission. Further, each submission is associated to a submitter by means of the submitter-ID, which is also the same as the submitter-ID used by the INSANE handing in the submission. Even though the submission-ID contains the original submission data, the submission date is stored separately, as it may happen that submissions are not immediately transmitted from the INSANE to the building server. As different types of submissions contain different data, the main submissions table should not contain the submission data. Rather, it should contain the type, while separate tables for each type contain the actual

[91] In Figure 6.2.15 the tables containing the original data are omitted for reasons of lucidity.
[92] It overrides any permission granted by the ACL negatively if a block is set.
[93] This shall ease implementation as the support of SSL is not always clearly indicated by the port used for communication.

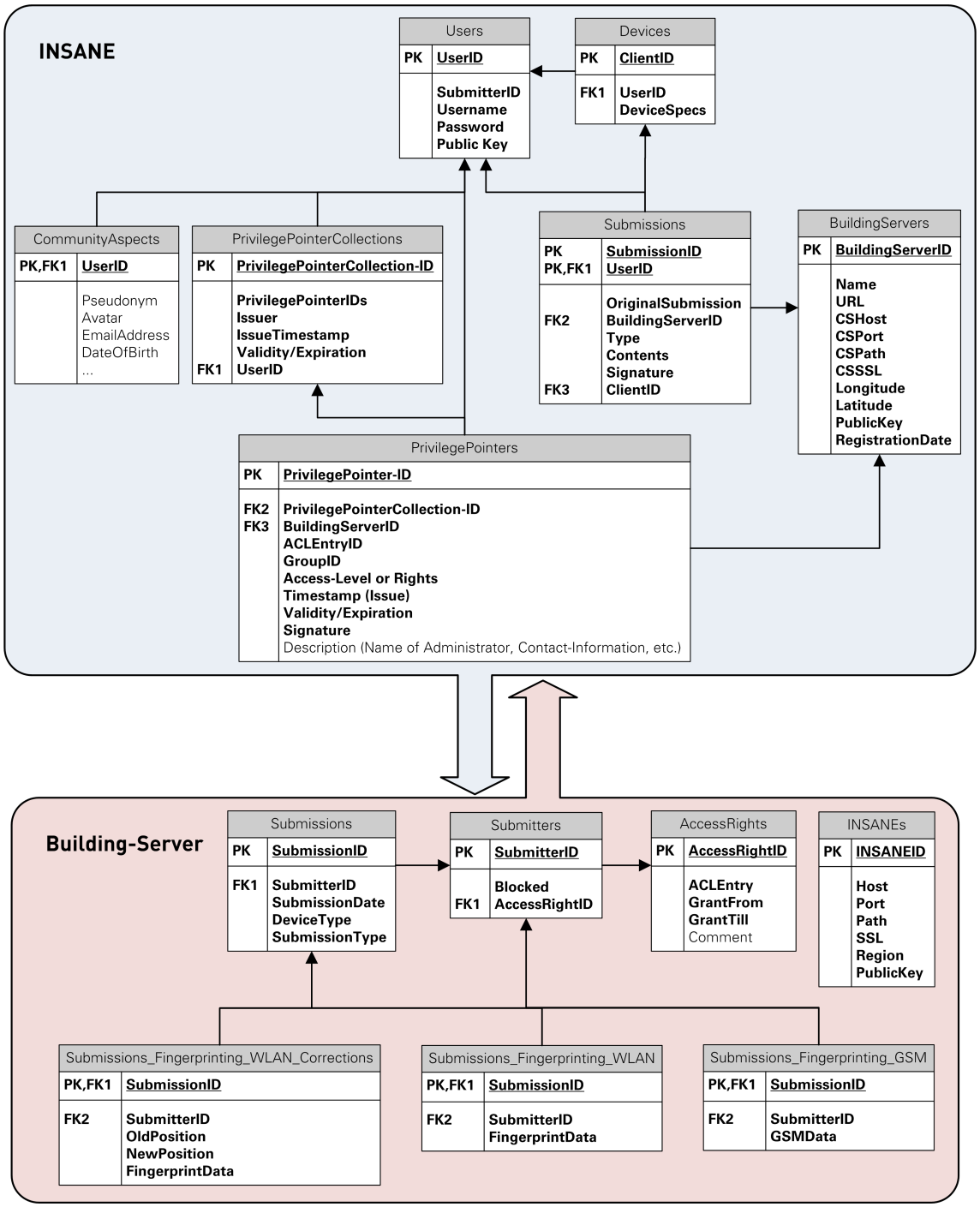


Figure 6.2.15: Conceptual design of MapBiquitous' crowdsourcing database structure

submissions. Allowing gathering of submissions made by similar devices or limiting to exactly one device type, an information on the device type used to create the submission is also stored.

The typed submission tables ('Submissions_Type') all follow the same schema. Identifying the submission via the submission-ID and associating it with a submitter via the submitter-ID, they store the submission data as present in the submissions without overhead[94]. Figure 6.2.15 contains three typed submission tables, one for GSM fingerprinting, one for WLAN fingerprinting and one for corrections to the positions delivered by the WLAN fingerprinting. As one can easily see, all three tables share the columns for the submission-ID and the submitter-ID, while the other columns differ.

On the INSANE's side, a set of seven tables is to be created, including the actual user data, community aspects, devices used, the submissions, the privilege pointer collections, a table on the privilege pointers with information on granted rights, and lastly a table on known building servers.

Before looking at the tables, a convention in anticipation of subsection 6.2.7 shall be made: In order to support distributed data storage, any ID used shall be created unique over the distribution.

The first of the tables, the user data table, is designed to be the central reference for user and client identification. As such, it contains a user-ID column as key, as well as further columns containing the username and password as defined in Definition 6.2.2 as well as Definition 6.2.4. Binding the user-account to the submitter-accounts on the building servers, the submitter-ID as defined in Definition 6.2.5 is also stored. Lastly, a column storing the public keys of the users/-clients[95] is earmarked, and used for the signature procedure described in subsection 6.2.5. – In accordance to the convention above, it shall be proposed to create the user-ID by calculating the initial username's SHA-256 hash. The so created distribution-wide unique user-ID could remain untouched, even when changing usernames later, or it could be changed as well. In the latter case, all references using this user-ID must be updated, as well, but it would prevent 'blocking' of user-IDs after usernames have been changed[96].

Assuming unique differentiation of users' clients is desired as proposed in subsection 6.2.1, the table for client-device data is mandatory. It shall be proposed such a table to contain a key based on the client-ID, as well as a column containing the corresponding user identification. An additional column for device specifications such as manufacturer, etc. can be allotted and should be considered, as it is impossible to foresee which information on the clients could prove to be of interest, later.

Next, the community-aspects table is designed to enable later interaction with social communities as described in subsection 6.2.1. For the very least, the user-ID should be designed to act as primary key as well as foreign key. All further columns actually depend on what type of community aspect is to be implemented. For example, pseudonyms, avatar images, e-mail addresses, date of birth, etc., could be stored.

Thirdly, the submissions table shall be introduced as a means to easily trace any submission made to any building server. As described earlier, for legal reasons it may be desirable to be able to trace all submissions back to the originating user. Hence, a special submission identification shall be proposed as key. With respect to the convention above, the submission identification should be distribution-wide unique. Hence, it shall be proposed to compute the submission

[94] Overhead is considered to be the submission-ID, the submitter-ID, the INSANE-ID of the INSANE handing in the submission, the device type and the signature over the entire data sent, which are all sent to the building server alongside the actually wanted submission data.

[95] At the time of this thesis' submission it has not been decided whether the PKI-usage shall be bound on user-level or client-level.

[96] Imagine, user 'Thomas' registers an account. The corresponding user-ID is `5dfcf9ef1fb1ecbce32fefe37ae99aff688-32a7e2ac74f52daa5a1bcd8038118`. Should 'Thomas' change their username while the user-ID maintains unmodified, future registration of another user wanting to use the username 'Thomas' would result in a user-ID collision.

identification over the timestamp of the submission, concatenated with the SHA-256 hash over the entire submission. For example, using the ISO 8601 'Zulu time-format' with the SHA-256 hash of the submission, a possible submission identification could be '`2012-08-31T17:27:35Z:8-feae6e9f88642b94937e2ee72d0aa4a9cb947ded00fd0207e93c328cb24dfd5`' for a submission made on 31 August 2012 at 7:27 PM and 35 seconds Central European Summer Time, where the contents yield the given SHA-256 hash. Further, the user-ID should be stored in the table, but not be transmitted to the building servers. But, the stored user-ID allows foreign key association with the submitter-ID in the user data table. This way, the table can be searched by submissions and by users. Further, the original submission – i.e. the variable-value-concatenation used by the GET/POST HTTP request without the submission – is stored alongside the signature, allowing alter authentification, confirmation and documented evidence of the submission. The contents of the submissions without the overhead created by the transmission[97] shall be stored. Allowing quick search for certain types of submissions, it is imperative to store the type of the submission in an dedicated column. Lastly, associating the submission with the device the submission originated on, a column is reserved for the client-ID.

Following the submissions table, information on all known building server is stored in a dedicated table ('BuildingServers'). Each building server is identified by its unique ID as well as their self proclaimed name. For reasons of searchability of building servers via the directory service, the name must be unique, too. Further, the contact information for HTTP-based communication are present in the table, yielding storage of the general access URL of the building server as well as the required information for crowdsourcing-access, meaning hostname, port and path to the crowdsourcing methods. Additionally, an information on whether the crowdsourcing methods require SSL access or not is stored[93]. For reasons of performance and reduction of directory service usage, the geographic information of the building servers is stored as well (by means of the longitude and latitude). Concluding the columns of the table, the registration date and public key of the building servers are stored.

As sixth addition, the privilege pointer collections table shall be used to summarise any user's privilege pointers. Beside the key functioning privilege pointer collection ID, the actual list of privilege pointers shall be stored. As the issuer of the collection may be of interest later[98], they shall be stored alongside the issuing timestamp and a validity or expiration. Lastly, the user-ID needs to be stored as well, otherwise the collections would not be allocatable to the users and their clients. – Once again, in accordance to the convention above, the used identification should be unique distribution-wide.

Finally, the table representing the privilege pointers shall be considered. Besides the key-functioning privilege pointer ID, which is used to identify the entries from the collections table's side, the affected building server's ID as well as the corresponding ACL entry with access level or rights should be stored. Should the building server pool submitters into groups, the group identification specified by the building server can be stored. Further, the signature of the privilege pointer created by the building server allows authenticity checks on the privilege pointer. The issuing timestamp alongside a validity or expiration needs to be stored in order to limit the validity of the privilege pointers[99]. Lastly, an additional descriptive column shall be reserved for comments, etc. – As true for all other identifications, the used identification used here should be unique distribution-wide in accordance to the convention above.

[97] Meaning without client-ID, password, etc. – Refer to the interface description in Appendix H.
[98] They are uninteresting for now, but the possibility of interest at a later time shall not be neglected.
[99] This does not affect revocability as building servers may revoke privilege pointers based on the privilege pointer ID at any time.

6.2.7 Optimisation towards Scalability: Distributed Hash Tables with DNS

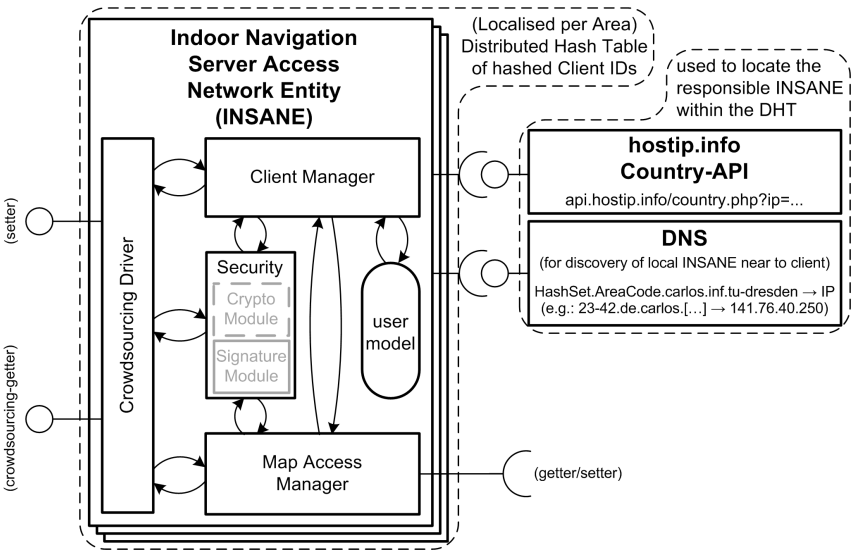


Figure 6.2.16: Extension to the INSANE: DHT

The INSANE as described in subsection 6.2.4 has one major disadvantage: scalability. Even though all building data are distributed over the distinct building servers, the INSANE as described above would centralise all crowdsourcing communication; hence, it would handle all clients, their data, as well as all submissions. Such a design is infeasible; therefore, the simple but effective solution of decentralising the INSANE into several INSANEs[100] should be pursued. – Now, rather than simply distributing/mirroring all data over several INSANEs and using a more or less random access strategy based on load balancing or so, an organised and well-defined access strategy shall be implemented. In order to achieve the well-defined access strategy, the use of a distributed hash table (*DHT*) for the organisation of the INSANEs shall be proposed. Herein, the term DHT should follow the classic definition of DHTs, meaning the pursue of specific design goals: autonomous decentralisation[101], fault tolerance[102] and scalability[103]. Without actually proposing to implement this herein presented optimisation to the INSANE component, the design should be prepared for a future extension in this direction; hence, the implementation of the INSANE (refer to chapter 7) must respect this.

Comment

> *Further information on how a DHT is supposed to function in a P2P networks (peer-to-peer networks) shall not be provided here as it would leave the scope of this thesis dramatically. The interested reader is referred to e.g. [BKK+03] or similar publications.*

The basic idea is very simple: any user accessing an INSANE is allotted to a corresponding hash area. The easiest way to accomplish this is by simply calculating the hash of the username. The exemplary interpretation of the hash as a hexadecimal number would allow the distribution of client accesses into distinct hexadecimal hash areas. These hash areas could be divided with growing number of INSANEs, e.g. dividing the area `[0-f]` (i.e. all usernames) could be equally divided into the hash areas `[0-7]` and `[8-f]`. Further, when reaching the boundary of 16 distinct hash areas, the hash areas could be subdivided, e.g. dividing the hash area `[7]` into the hash areas `[70-77]` and `[78-7f]`. Using the 256 Bit SHA-256 hash algorithm would allow a maximum of $2^{255} \approx 5.79 \cdot 10^{76}$ hash areas[104], more than enough to cover all uniquely distinct usernames that have existed, are existing and will ever exist in a reasonable scope of time.

[100] Even though the correct plural would be 'Indoor Navigation Server Access Network Entit***ie***s', the more intuitive 'INSANEs', maintaining the 'y', shall be used.
[101] The INSANEs collectively form the INSANE distribution without any central coordination.
[102] The INSANE distribution is reliable even with distinct INSANEs joining, leaving, and/or failing the distribution.
[103] The INSANE distribution functions efficiently even with numerous clients, INSANEs and building servers.
[104] The author of this thesis assumes that the term 'area' is only valid when at least two values are included in the areas. 2^{256} would be the maximum amount of usernames supportable by the INSANE distribution.

In order to have the INSANE distribution be able to quicker react to the fault of a single INSANE, it shall be proposed that each INSANE be able to handle the hash area for its left/lower and right/upper neighbour. With one, two or three INSANEs this leads to a very silly distribution with all of them handling the hash area `[0-f]`. But, as soon as a fourth INSANE joins the DHT and the same copy-strategy is followed, after the redistribution time the distribution changes to:

- INSANE 1 is responsible for hash area `[0-3]`, thus handles the hash area `[d-7]`,
- INSANE 2 is responsible for hash area `[4-7]`, thus handles the hash area `[0-b]`,
- INSANE 3 is responsible for hash area `[8-b]`, thus handles the hash area `[4-f]`, and
- INSANE 4 is responsible for hash area `[c-f]`, thus handles the hash area `[8-3]`.

The basic idea is depicted in Figure 6.2.17, where a user (`PeterMustermann`) finds the primary INSANE responsible for the hash area (`[9-b]`) in which the username (hashed identity `bbe29c-00e7661de0fd6ab795f49bda549ffcae5dd63d96d8670b712360528c32`) belongs, as well as both backup INSANEs. Of course, proper interfaces need to be present on each INSANE in order to guarantee proper DHT-handling; therefore, the importance shall be emphasised.

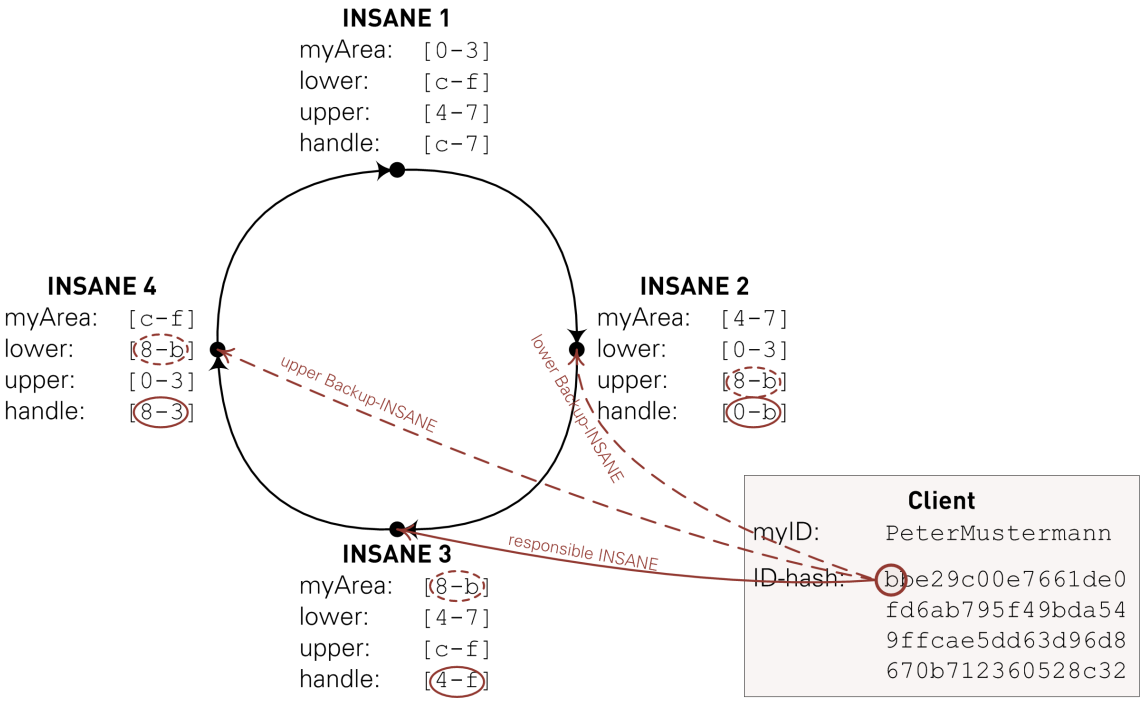


Figure 6.2.17: Exemplary INSANE distribution with four hash areas; each username has one primary INSANE and two backups

Now that the basic advantages of DHT-usage have been pointed out, a major disadvantage for the use within MapBiquitous shall be pointed out: the distribution does not pay any respect to the users' geographic positions. – It is desirable to reduce communication distance, hence automatically reducing the communication delay.

The optimisation proposed up to now does not pay any respect to the geographic distribution of the users. Hence, it is possible that a user accessing the INSANE distribution from San Francisco, CA (United States of America) finds their responsible INSANE to be an INSANE in Dresden, SN (Federal Republic of Germany). Even when considering the distance only as the crow flies, a total of 9264km must be covered; with an optimistic nominal velocity of propagation ($0.8c$) and a package size of 1 Bit, a total of 77.25ms pass before a response is received. But, as communication lines are not placed following a straight line as the crow flies and transatlantic cables supposedly only provide a nominal velocity of propagation of $0.6c$, a more realistic value would be $2 \times \left((9264096\text{m} \times 1.95) \div 194856100\frac{\text{m}}{\text{s}}\right) \approx 0.18542\text{s} = 185.42\text{ms}$. This value can be

easily reproduced when using a personal computer in Dresden and sending a 'ping' request to the IPv4-address 169.229.216.200 (DNS-entry 'berkeley.edu'). Now, if taking into consideration that a user is most likely interested in data for a building near their location, and further assuming that the responsible building server is also located there, the delay increases even further to 144.5ms in the optimistic case, and totally unacceptable 370.84ms if we consider the realistic[105] numbers. – Obviously, a solution needs to be provided for the geographic localisation problem.

When considering that IP-addresses are distributed following a well-defined geographic schema by the Internet Assigned Numbers Authority (IANA) and its regional internet registries (RIR), namely the African Network Information Centre (AfriNIC), the Asia-Pacific Network Information Centre (APNIC), the American Registry for Internet Numbers (ARIN), the Latin America and Caribbean Network Information Centre (LACNIC) and the Réseaux IP Européens Network Coordination Centre (RIPE NCC)[106], it would make sense to use the accessing client's IP-address to determine their geographic location[107]. For this purpose the country-API of *hostip.info* can be used, which currently supports the recognition of 247 country-locations[108] based on IP-addresses, and is – in contrast to similar APIs of the RIRs – free to use, even when conducting consecutive queries in a row.

Finally, when having acquired the geographic location of a client based on its IP-address, it makes sense to select an INSANE based on this geographic information. Therefore, it shall be proposed to divide the above suggested INSANE distribution into regional hash area distributions. The contents of the regional distributions should be synchronised regularly, as different INSANEs in different regions of the world will definitely handle the same hash areas. Further, it shall be proposed to extend the now divided INSANE distribution with a DNS interface, registering distribution heads[109] that allow easy finding of the next nearest responsible INSANE handling a username's hash area corresponding to the accessing client's user. The basic idea is depicted in Figure 6.2.18.

The differently coloured arrows in Figure 6.2.18 represent different types of communication. Red indicates communication from the client to an INSANE, orange the corresponding reply, purple a query to the country-API of hostip.info, grey the corresponding reply, green a DNS query, and blue the corresponding DNS-reply. – The example in Figure 6.2.18 presents a client (hashed identifier `bbe...`) in the United States of America trying to access INSANE X in the Federal Republic of Germany in step 1. INSANE X does handle the corresponding hash area (`[a-b]`), but recognises that the client is not in the same geographic region after utilising hostip.info's country-API in step 2 and receiving the region information 'US' in step 3. INSANE X then retrieves the regional distribution head in the United States of America utilising the DNS in steps 4 and 5. Rather than replying to the initial request of the client with a confirmation or results to the corresponding initial request in step 6, INSANE X replies with the connection information of the regional distribution head in the United States of America, INSANE A. The client then repeats its original request in step 7, this time directed at INSANE A. As the hash area of INSANE A does not cover the hashed identifier of the client, INSANE A directly replies with the corresponding connection information of INSANE Z in step 8. Finally in step 9, the initial request of the client is repeated once more, this time directed at the correct INSANE handling the corresponding hash area in the geographic region of the client.

Lastly it should be mentioned, that the proposed concept should have the initially contacted INSANE handle the client's request iff the contacted INSANE is the responsible INSANE in the client's geographic region handling the hash area containing the hashed username of the client's

[105]However realistic it may be when actually limiting the package length to 1 Bit as described earlier...

[106]The Réseaux IP Européens Network Coordination Centre (RIPE NCC) is responsible for the IPv4 class-B address ranges 141.30.0.0-141.30.255.255 and 141.76.0.0-141.76.255.255 which the Technical University of Dresden uses.

[107]The responsible RIR of an IP can be determined by a corresponding entry in a database provided by the IANA, e.g. `http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml` – Accessed 20. August 2012

[108]`http://www.hostip.info/bulk/countries.html` – Accessed 20 August 2012

[109]However this shall be implemented, it is important that each region is represented by at least one permanently reachable INSANE.

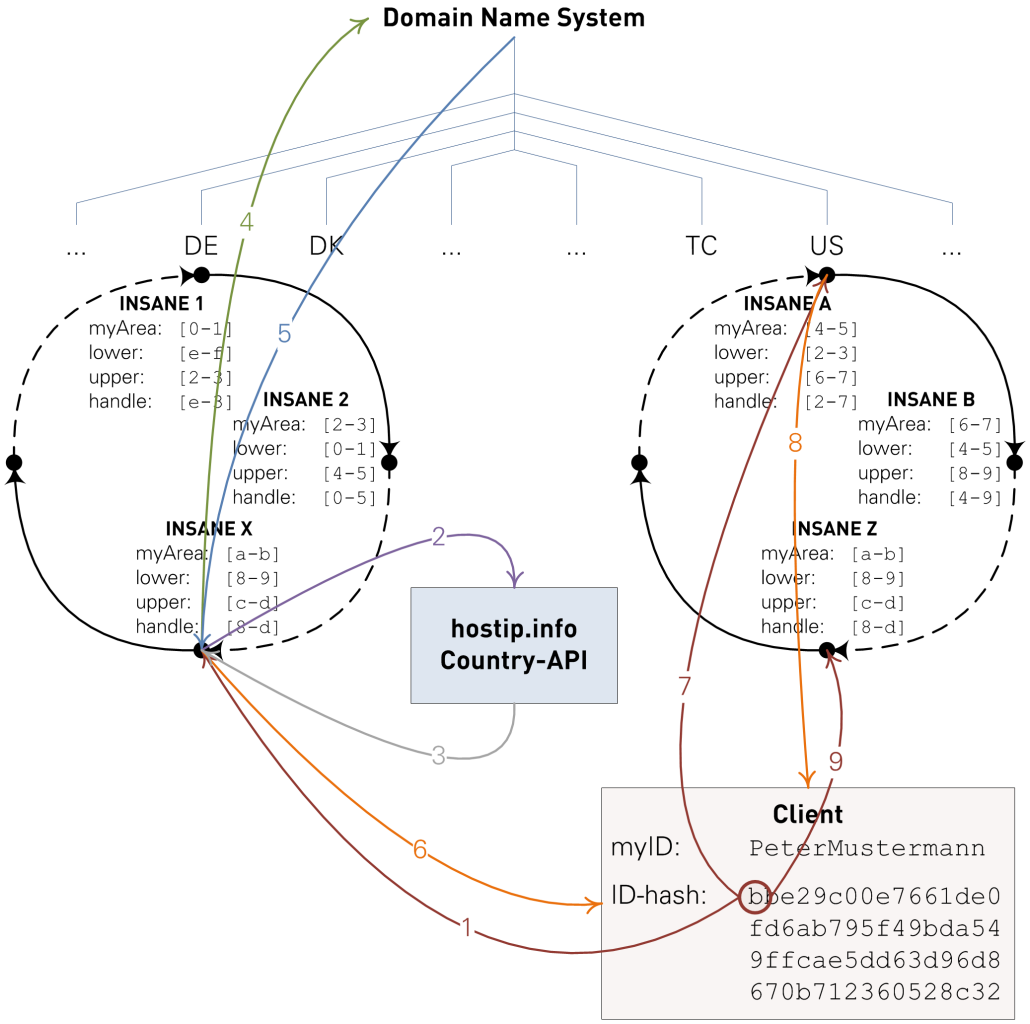


Figure 6.2.18: Exemplary INSANE distribution extended with an DNS interface

user, or iff the contacted INSANE was not able to retrieve the responsible INSANE via hostip.info's country-API and/or DNS, or iff the DNS query results in the information that there is no INSANE in the geographic region of the client. – This behaviour can of course be optimised by determining the geographic region containing an INSANE closest to the client's geographic region, but at this time the author has no conclusive concept of how to design/implement such an extended DNS query, except for using DNS' GeoIP-API, which would bloat the entire design.

6.2.8 Summary

A new component, the Indoor Navigation Server Access Network Entity (INSANE), is added to the MapBiquitous system in order to handle all crowdsourcing related communication between the system's clients and building servers. As such proxy, the INSANE shall guarantee anonymity of the system's users and their clients towards the building servers. Hence, encryption and anonymisation are key aspects of the INSANE's operation. Corresponding required modifications to the building servers are limited to submissions handling and ACL management, while modifications to the clients require changeover of established interfaces as well as creation of new interfaces.

As a tribute to scalability, amendments of the INSANE in shape of a distributed hash table extended by the Domain Name System contribute to user access handling while paying respect to the regional distribution of the users and INSANEs within the MapBiquitous system.

All of the concepts and ideas introduced in this section are summarised in Figure 6.2.19, which – in contrast to Figure 6.2.1 – displays the desired state of the MapBiquitous system by the time this thesis is going to be handed in for review.

6.3 INTERPLAY OF CLIENT AND SERVER

The interplay of the crowdsourcing client as introduced in section 6.1 and the crowdfunding server as introduced in section 6.2 shall be expected to be free of malicious intent. Hence, an optimistic communication and interplay approach shall be followed. The reason for this conceptual design decision shall be depicted in the following example.

When present in a building, the client normally needs to load the ichnography, the layers as well as POIs and WLAN-fingerprints. Should there be any information the user deems wrong, they would suggest corrections. This is the basic idea of the crowdsourcing concept. Additionally, the user expects to be informed whether their submission was accepted so that the same correction will not be resubmitted; ideally, corrections are displayed in the building plan. There are three ways to achieve this. Firstly, the client could store any corrections in its local storage and replace outdated data from the server with its own corrections. As soon as the server has updated its data, the client requires no more replacement. Unfortunately, this concept does not work, as the server could either correct data, or add new data. Should there be several submissions from different users concerning different information, it is impossible for the client to determine which data is actually updated and corresponds to the stored submissions. So this solution is not feasible. Secondly, the client could ask the INSANE as the proxy that handled the submission towards the building server to help by gathering all submissions of the client and querying the building server what intermediate results the submissions have triggered. Unfortunately, an anonymity problem arises, as the initial direct loading of data from the building server inevitably leads to exposing the clients IP-address to the building server. As soon as the INSANE then queries all submissions of the client, which as a reminder were intended to be anonymous (refer to subsection 6.2.5), the building server can link the IP to the submissions due to the temporal correlation of the requests. Should the client be in the situation of using the

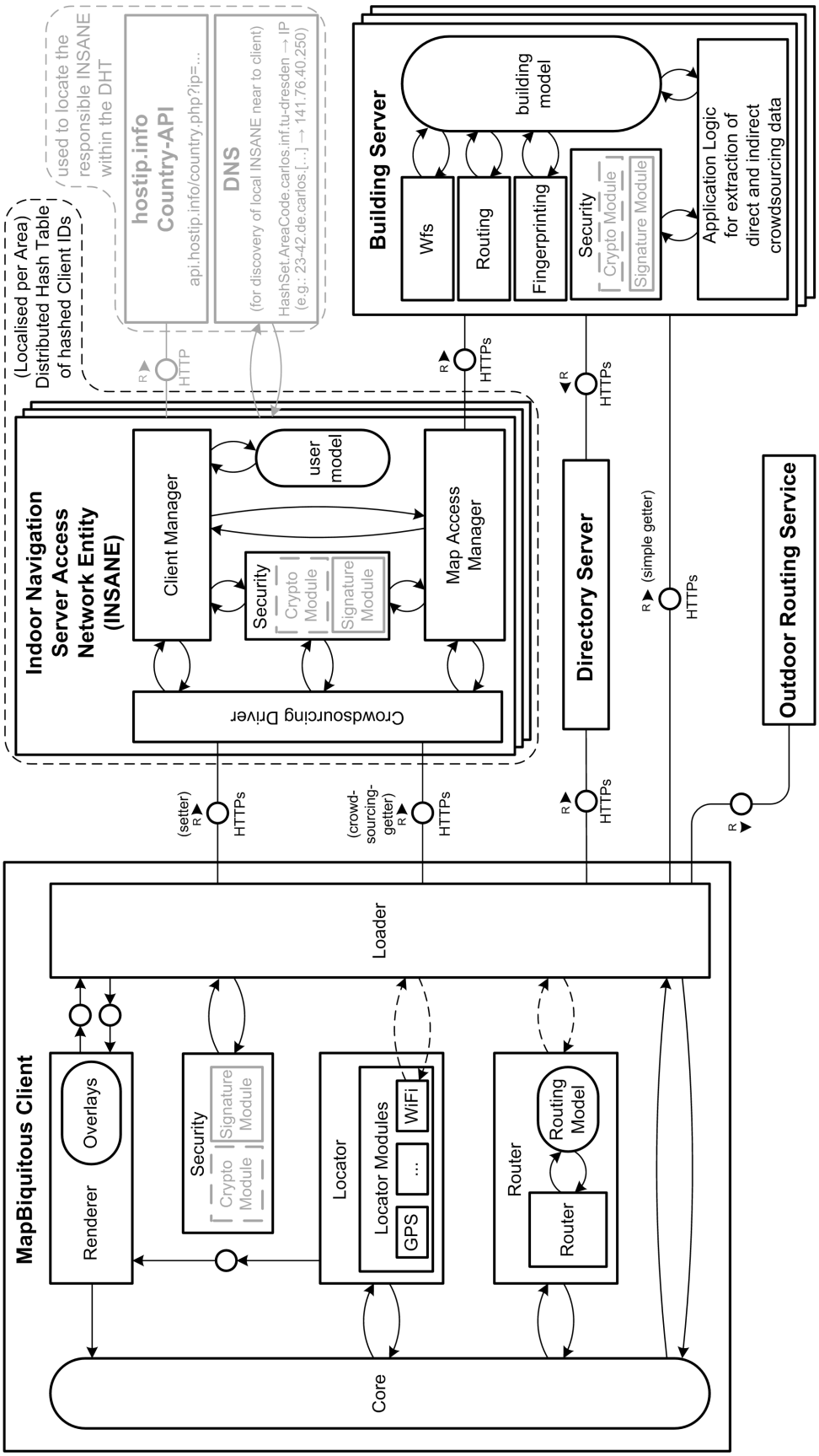


Figure 6.2.19: Conceptual design of MapBiquitous' crowdsourcing architecture as planed by 15 October 2012; the greyed components are prepared and work as stubs, but remain to be implemented with proper functionality

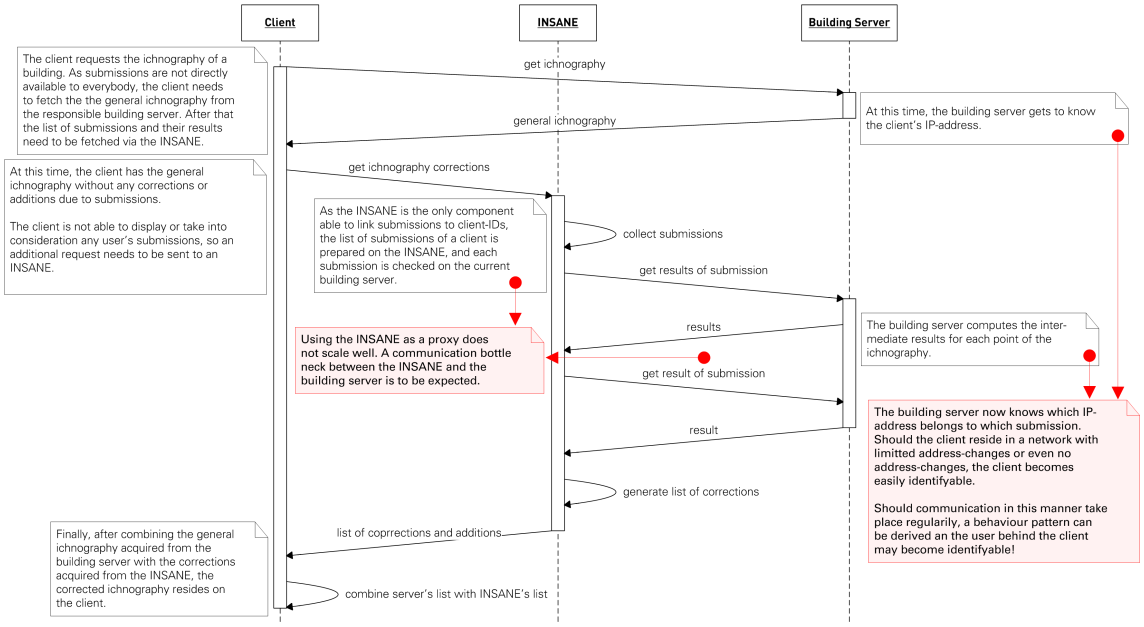


Figure 6.3.1: An exemplary sequence leading to de-anonymisation as well as a communication bottleneck

same IP over longer periods of time, e.g. with a long DHCP lease time or even static IP, usage pattern analysis may lead to identification of the user behind the client. Besides this anonymity problem, a communication bottleneck is to be expected at the interface between the INSANE and the building server. The problems with the second approach are depicted in Figure 6.3.1.

The third solution envisages that all communication is proxied via an INSANE for crowdsourcing-participants and that the building servers only return a submitters own contributions as long as they have not been sifted, i.e. integrated into the general dataset. This way neither the deanonymisation problem nor the communication bottleneck occur. Further, any client could be allowed to user the proxied communication path via INSANEs in order to conceal their IP-address from the building servers. The concept is depicted in Figure 6.3.2.

Comparing them, the third solutions is the logical choice for implementation. Having this decided, a closer look at how clients, INSANEs and building servers communicate is profitable. Using HTTP-based communication limited to POST and GET requests allows usage of existing infrastructures while at the same time no special preparations to firewalls and other network components are required[110]. Taking this idea even a level lower, usage of XML or other special description languages can be limited to actual content, e.g. a WFS-reply. All other information, especially on success or failure of method calls, can be communicated by the means of HTTP itself, i.e. by RFC 2616 (Section 10) status codes, such as 'HTTP/1.1 202 Accepted', 'HTTP/1.1 404 Not Found', etc. Using such low level means of communication, the focus should be shifted to POST requests, as they transport variables within the HTTP header in contrast to within the URL for GET requests. As encryption is required, encrypting a GET request with the means of HTTPs would be meaningless as the variables and their contents remain in plain-text in the URL. Wanting to encrypt them would require a crafty URL-encryption. In contrast, encrypting POST requests with the means of HTTPs is actually very simple: no additional effort is required as HTTPs encrypts the header alongside the HTTP packet's body.

Analysing the aspect of authenticated communication, the idea of using signatures for the requests and replies seems rather simple, but it is effective. But, one important condition must be met: all participants must agree on which information is to be signed, and in which way. For this, a very simple communication protocol shall be defined, which can be found in the following subsection.

[110] Of course, this is only true as long as the clients, INSANEs and building servers refrain from using exotic ports.

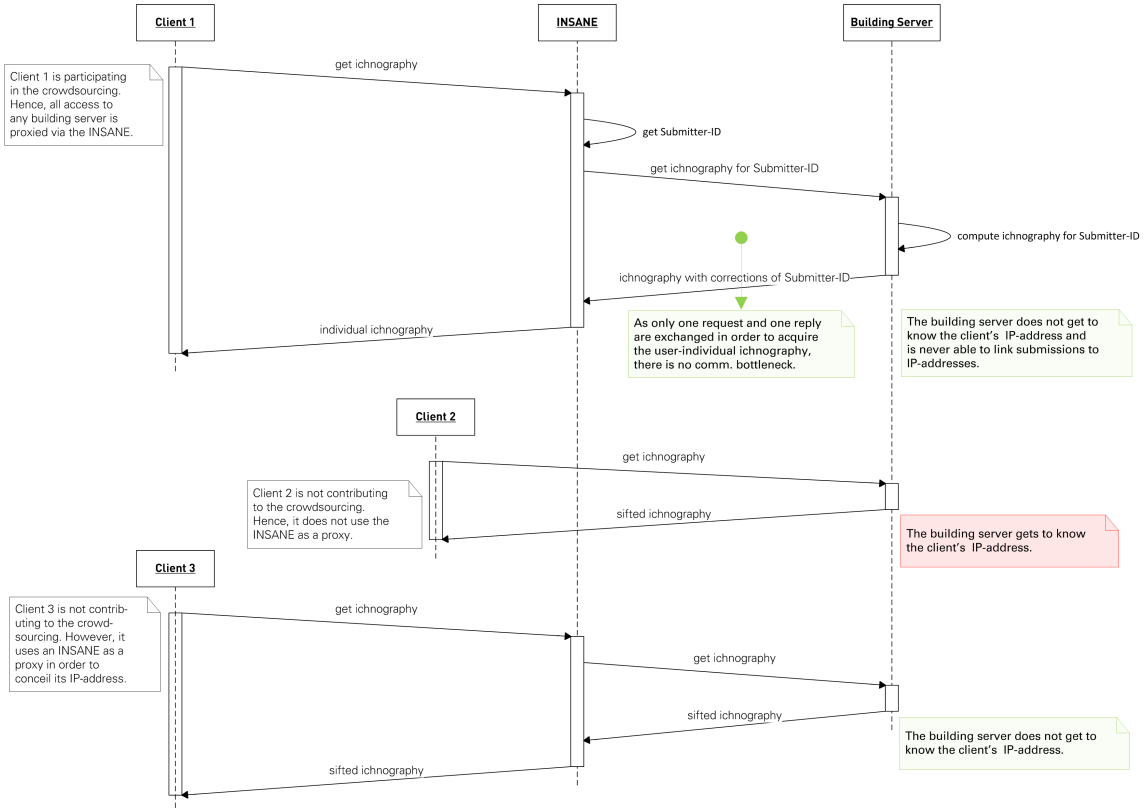


Figure 6.3.2: An exemplary sequence solving the de-anonymisation and communication bottleneck problems

6.3.1 Extensible MapBiquitous Crowdsourcing Communication Protocol

The communication between MapBiquitous' clients, INSANEs, building servers and directory servers is standard HTTP communication, either encrypted or not. Only a subset of the status codes defined in RFC 2616 (Section 10) is allowed, completed by status code `424` from RFC 4918 (Section 11). Even though the denotation of the status codes is basically the same, the status codes are returned under deviating conditions:

- `HTTP/1.1 200 OK`
  Successful method call with expected results. The body of the HTTP packet may be empty or contain the actual result.
- `HTTP/1.1 201 Created`
  Successful method call with expected results. The body of the HTTP packet should be empty, but may contain feedback.
- `HTTP/1.1 202 Accepted`
  Successful method call. The submitted data was accepted, but processing has not finished, yet. The body of the HTTP packet should be empty, but may contain feedback.
- `HTTP/1.1 302 Found`
  The resource in demand was found. The body of the HTTP packet must contain the location of the resource.
- `HTTP/1.1 303 See Other`
  The request must be repeated on another host. The body of the HTTP packet must contain either a reason or the location of the proper host.
- `HTTP/1.1 400 Bad Request`
  The method call was unsuccessful due to malformed or incomplete variables, or the client and host use different versions of the extensible MapBiquitous crowdsourcing communication protocol. The body of the HTTP packet may be empty or contain error descriptions. A 400 error must be dealt with by the client, not the host.

- `HTTP/1.1 403 Forbidden`
  Execution of the method call was refused due to insufficient access right, an invalid signature or execution of the method would overwrite an existing resource. The body of the HTTP packet may be empty or contain error descriptions.
- `HTTP/1.1 404 Not Found`
  The resource in demand was not found or the execution of the method returned an empty result. The body of the HTTP packet should be empty, but may contain error descriptions or the location of a host that may be able to return a different result.
- `HTTP/1.1 409 Conflict`
  The method call ended with the host being in a state of conflict. This status may only be returned iff the resource or the computations result is ambivalent, i.e. the host has to (randomly) select one of them for the reply. The body of the HTTP packet should contain exactly one of the possible results.
- `HTTP/1.1 424 Failed Dependency`
  Execution of the method call was denied due to unfulfilled dependencies. This status must be returned iff the method has preconditions that need to be met, e.g. registration of the client before calling the method on the host. The body of the HTTP packet may be empty or contain error descriptions.
- `HTTP/1.1 500 Internal Server Error`
  Procedure call or execution of the method call resulted in an unrecoverable error on the host. This error must be returned iff the host encounters missing modules, code errors, database errors or obviously malformed data. The body of the HTTP packet must contain information on the error. Should the error be expected to last longer, an estimate on the recovery time should be returned, as well.
- `HTTP/1.1 501 Not Implemented`
  The method call could not be handled since the method is not implemented on the host. This status must be returned iff the method is implemented as a stub on the host. The body of the HTTP packet may be empty or contain the location of a host that may be able to execute the method call.
- `HTTP/1.1 502 Bad Gateway`
  Execution of the method call depends on communication to another entity. However, the host encountered a communication error when contacting the entity. The body of the HTTP packet should contain an error description. Additionally, information on the unreachable/erroneous entity may be provided.
- `HTTP/1.1 503 Service Unavailable`
  The method call could not be handled since the method does not exist on the host. The body of the HTTP packet should be empty, but it may contain error descriptions.

Besides the status codes, additional header fields are stipulated:

- `Timestamp`
  The time the packet and the corresponding signature were created. The timestamp must be a ISO 8601 (Section 4.3.2) UTC timestamp ('Zulu'-format), e.g. '`2012-10-06T16:17:14Z`'.
- `Signature`
  The signature over the body of the HTTP packet and the '`Timestamp`' header field. This header field is only to be used for replies, never for a request.
- `Result base`
  The base of the method execution result. Its value may be either 'anonymous' for a general result or a submitter-ID for a personalised result. This header field is only to be set by INSANEs and building servers.
- `Result correction`
  The submission(s) a result deviating from the general result (available to everybody) is based on. Its value must be a comma-separated list of submission-IDs or exactly on submission-ID. This header field is only to be set by INSANEs and building servers and it may only be set iff the '`Result base`' header field is set.

The body of the HTTP packet transmitted may contain any data that is feasible for HTTP communication, especially plain-text or XML data.

Several time now, signatures were mentioned. These signatures shall authenticate/validate the contents of the HTTP packets and make modifications and/or unauthorised packets recognisable. The signature should be calculated:

- *Requests*:
  Over all POST/GET variables except for the signature itself. For this, all variables and their values are concatenated using the default HTTP variable header-notation, e.g. '`variable1=value1&variable2=value2&...`'
  The order of the variables is imperative; therefore, this communication protocol demands that client and host use the same order when calculating the signature. The order must be defined at implementation time.
- *Replies*:
  Over the entire body of the HTTP packet concatenated with the '`Timestamp`' header field similar to the default HTTP variable header-notation, but allowing two or more consecutive line breaks, e.g. '`result=anXMLentitiy&timestamp=2012-10-06T16:41:22Z`'
  The order is imperative, demanding for the result to always be placed before the timestamp.

While the signatures calculated for requests are to be added to the GET/POST request variables as '`&signature=`' followed by the signature[111], the signatures calculated for replies are to be added to the header using the above mentioned '`Signature`' header field.

6.4 CONCLUSION

The crowdsourcing architecture designed follows the intuitive approach of separating the crowd and the crowdfunders, dividing the architecture into a client-side and a server-side. This is compatible with the existing MapBiquitous architecture as it is divided in the same way. Effort was focused on a distributed, reliable and scalable architecture that supports the existing functions as well as new functions (i.e. crowdsourcing). For this, the client and server are extended by new crowdsourcing modules. Additionally a proxy – the INSANE – is introduced into the architecture, basically relaying all crowdsourcing-related communication between the clients and the building servers. In standard networks, the architecture's design requires no modifications to firewalls, etc. as the entire communication is HTTP-based, using mainly POST and a few GET requests via HTTP default ports 80 (without encryption) and 443 (with encryption). Further, only standard HTTP 1.1 status codes as defined in RFC 2616 (Section 10) and RFC 4918 (Section 11) are used. The building server component as well as the INSANE are designed to be extensible by being modularised. By this, the architecture requires no modification for different types of crowdsourced information, as it is sufficient to simply deploy additional communication and application logic modules into the INSANEs and building servers, as well as creating new database tables into the existing building server databases for each newly to be supported type of crowdsourcing. Further, the proxy-based communication is decoupled from single network instances as the architecture envisages at least threefold redundancy. From the client's vantage point, the remaining single point of failure remains to be the building server. Even though not discussed in this conceptual chapter, this problem can be solved by introducing redundancy to the building servers, too.

[111] Note that GET as well as POST use the same variable notation. They only differ in the placement of the variables, having them in the URL for GET, and in the header for POST.

7 PROOF OF CONCEPT WITHIN MAPBIQUITOUS

„Wer nicht genau weiß, wohin er will, braucht sich nicht zu wundern, wenn er ganz wo anders ankommt."

Rober F. Mager

PROOF OF CONCEPT WITHIN MAPBIQUITOUS

Comment

> *Before actually starting into this chapter, it shall be noted that the aspects considered within this chapter are limited to the INSANE and building server crowdsourcing module and their inclining system modifications. Any aspects corresponding to the client and/or modifications of the same will not be given herein. The interested reader is requested to kindly refer to [Bom12].*

Sometimes one finds the actual implementation to derive from the documentation or accompanying papers, theses, etc. Sadly, this is also the case for the MapBiquitous project. There exists a variety of assignment papers and theses dealing with the MapBiquitous project, each one of them praising their concept and the accompanying implementation. This does not surprise; of course a student is keen to present their own work in the best light in order to reach the best possible rating. Unfortunately, the reality of implementations proves to deviate from the highly praised theory. Exemplarily, the implementation of the fingerprinting server shall be analysed: The corresponding concept in the assignment paper claims the presence of a fingerprinting module on each building server [Gru12], while in rality, there is one fingerprinting service running for all building servers. Even the concept of several building servers is not implemented, as one server[112] simulates the existence of diverse building servers, while they actually are only one variable to be set when accessing the web feature service (WFS) of the running geoserver. Nevertheless, this still allows the simulation of several building servers using different access URLs[113], while the mentioned fingerprinting service handles fingerprints on a global level, only storing in which building the fingerprint was generated.

As the author does not wish to run riot by ranting over the unfortunate deviation of concept and implementation, a short but far from exhaustive list of problems shall be presented before showing up how problems were solved or circumvented. Following that, a review of deviations from the concept developed in section 6.2 shall be presented, concluding with an architecture diagram of the actual implementation. In this spirit, the author of this thesis is not superior to their predecessors, but at least the author wishes to have the courage to point out his own mistakes, enabling potential successors an easier access to 'the reality of MapBiquitous'.

7.1 PROBLEMS

In order to have an approach to the diverse problems that occurred during the creation of this thesis, the problems shall be presented sorted by their impact on the implementation in ascending order.

First, the coordination of a team is always challenging, even if the team only consists of two members and a supervisor. Luckily, the cooperation of the authors of this thesis and [Bom12] was fruitful. Nevertheless, the definition of the required interfaces between client side and server side proved to be challenging. Especially, the architecture concept had to be revised several times when discussing interfaces, as use cases had to be modified after discovering an error, etc. As an example, the abandoning of total anonymity of the clients toward the building servers and replacing it with an identity protection mechanism just one week before the evaluation and three weeks before the submission of this thesis was very challenging.

Next, the shortened processing time of this thesis lead to a submission deadline in the middle of October, which is very unpleasant for exhaustive user tests in the group of students, as such

[112] CAmpus Range LOcation System – `https://carlos.inf.tu-dresden.de`
[113] Differing in the mentioned variable.

test would have to be conducted at the end of September in order to allow evaluation before the submission of the thesis. Nevertheless, this problem could be shifted to the client side, making it challenging for Gerd Bombach, while the server side evaluation could focussed on scalability and performance aspects.

The next problem originated in the use of the Apache Maven Eclipse plugin, leading to unusability of the Eclipse development framework. As the fingerprinting service explicitly required Apache Maven, modifications to the fingerprinting service were not possible without rendering all other projects in the workspace unusable. The obvious solution of using separate workspaces was not applicable as modifications to Eclipse itself were required. Hence, two separate installations of Eclipse would have been required, which proved to be infeasible with respect to available resources and time required to handle them on separate devices.

The database structures used proved to be very challenging. It took little over a month until the capabilities of the used Postgres database with respect to automated coordinate transformations were clear. Additionally, the mapping of WGS-84 coordinates to SRID-4326 was neither documented nor hinted as default behaviour. After having figured this out, parts of the dataset presented themselves as not obliged to this mapping, using different mappings, making inter dataset transitions not intuitively possible. In detail, the database 'MapBiquitous' uses SRID-4326 only for the 'POI' table, while 'point_of_interest' table uses SRID-'-1', 'MapBiquitous-Neu''s tables all use SRID-'-1' with the exception of the 'POI' table which uses SRID-4326, and 'MapBiquitous-Navigation''s table also mainly use SRID-4326 with the exception of the 'Mainentrance' table which uses SRID-'-1'.

The previous problem automatically leads to one of the most difficult to handle problems: There existed a variety of databases and tables, of which is not actually clear which is used and which not. For example, the 'POI' table of the 'MapBiquitous' database provided point of interest data, whereas all other geoserver data were provided by the 'MapBiquitous-Neu' database. Further, the databases themselves were not clearly structured as provided in the corresponding concepts. Basically, one Postgres database provided for all building servers. In the end, two month (!) had to be invested into separating the entries of all tables into distinct building server databases, which of course, still run on the same database server. But, at least now the building servers' data are logically separated within the database server.

In conjunction with the efforts mentioned in the (partial) solution of the previous problem, the used GeoServer 2.1.3[114] denied any form of easy modification. Neither could the features be redefined properly even when the student who initially set up the server tried to help nor could the underlying data structures be modified (i.e. in the database directly) without rendering the geoserver unusable. Further, the server had and still has an irreproducible memory consumption problem, rendering the server unreachable after some time, sometime crashing the entire underlying Tomcat server, or once even the entire virtual machine within which the CARLOS server operates. Sadly, this problem could not be solved; only the time between consecutive crashes could be reduced by removing unnecessary components from the virtual machine and enlarging Tomcat's memory stack.

For nearly two and a half month the source codes of some server components were not available, pushing the actual review of existing code wide into the conception of the crowdsourcing architecture. When the sources became available and were finally reviewed, modifications to the conceived crowdsourcing concept were inevitable in order to pay respect to components that could either not be modified in the limited time in order to adhere to required architecture modification, or remain unmodifiable due to missing sources. Unfortunately, after some time an important part of previous documentation, namely the Wiki containing information about MapBiquitous' second iteration, ceased to be reachable or simply did not exist any longer; thus, some information turned out to be no longer accessible.

[114] The used GeoServer project is a transactional J2EE-implementation of the OGC Web Feature Server specification and the OGC Web Coverage Server specification.

Server components ought to be deprecated proved to be actually still in use. The MS4W server on the virtualised CARLOS shall provide as an example. While MS4W was used in MapBiquitous' second iteration, the current third iteration does not use MS4W. At least that is what the concept states, so the MS4W server should be removable from the virtual machine. Nevertheless, after deactivating the MS4W server the fingerprinting service and parts of the geoserver functionality unexpectedly terminated. A code review (as far as possible) and extensive search in the old documentation of MapBiquitous' second iteration finally hinted that parts of MS4W were still required for CGI calls, etc. This behaviour was not documented in the concept of MapBiquitous' third iteration. In order to save system resources, the deactivation and removal of the MS4W server was desirable, especially as the proof-of-concept implementations of the INSANE and the building server crowdsourcing modules were supposed to be deployed into a very basic Apache httpd server. In the end, these deployments were conducted into the running MS4W server's httpd service. In this spirit, this problem is still not solved, only circumvented.

Finally, the last and most pressing problem in the end proved to be time. This is not to be mistaken with the time problem arisen due to the shortened processing time. The definition of the required crowdsourcing taxonomy as well as the concept of the crowdsourced MapBiquitous were conceived within one month, and implementation was ready to go. But, due to the sum of all previously mentioned problems, a guerrilla warfare against smaller and larger problems had to be conducted, slowly shifting the focus from crowdsourcing itself to providing a crowdsourcing architecture and finally to fixing all kinds of existing issues. In the end, the three month planned for implementation were fully filled with issues, to which the implementation came on top, reducing testing and evaluation time to a minimum of three weeks by the time these lines are being written. It is foreseeable, that the finalisation of this thesis will overlap with the evaluation in the last week before the thesis' submission. Sadly, in retrospective the author sees no way how the time could have been planned more efficiently, except by having all unnecessary problems (missing source codes, database structure problems, implementations deviating from concepts, etc.) not occur. Regrettably, those problems could not be influenced.

7.2 DEVIATIONS FROM THE CONCEPT

The most pressing deviation from the concept conceived within this thesis definitely is the use of only one actual server pretending to be several building servers. Further, this same server is used for the deployment of at least one INSANE instance, while the concept clearly sees INSANEs and building servers on separate servers. Lacking the resources to set up separate physical or virtual machines for each architecture component, the building servers had to remain on 'CARLOS', but that is not a problem. In fact, maintainers of several buildings should be allowed to share the same physical or virtual server in order to save resources. It is only imperative for the different building server datasets to be accessible under clearly distinct access interfaces; in the simplest case differing URLs. The no-go in the described deviation is the presence of an INSANE on the building server. The concept clearly states anonymity of clients towards the building server as a design goal (goal 2d), but having the INSANE and a building server on the same physical or virtual machine undercuts anonymity.

The concept also sees the directory server unmodified as any INSANE should – iff necessary – use the default functional range of the directory server. Nevertheless, the definition of the interfaces as well as the corresponding implementation of the INSANE's functionality has proven the need for INSANEs to be able to lookup building servers not by their geographic location[115], but also by their identity/alias. Hence, the directory server was modified and amended with an alias search interface. The same applies for the reverse case of building servers having have to look up an INSANE. Further, it should be noted that the original directory server could not be easily amended with the additional functionality due to unconceivable errors in the original source

[115] The WGS-84 coordinate, or to be more precise, their location between minimum and maximum longitudes and latitudes.

code; hence, a replacement directory server implementing the original and the new search capabilities was set up. However, the replacement lacks all setter interfaces, e.g. it is not possible to modify access URLS, etc. For those functions, the original directory server is still available and can be used, as both directory servers share the same database.

With respect to security, the concept envisages usage of HTTPs for all communication between the clients, INSANEs and building servers. Due to problems with HTTPs-implementation, especially HTTPs connection handling in Android and certificate management in PHP, HTTPs is deactivated even though supported on server side. Hence, the communication is unencrypted in the moment.

As the concept was conceived before the majority of the problems described in the previous section occurred, the concept envisages a clearly modularised building server. The implementation reality still forces a dedicated fingerprinting service and web feature service handling all buildings. Solely the routing and crowdsourcing components are actually deployed per building. In the end, the building server crowdsourcing module (BSCSM) was introduced into the architecture as a new component. This component is designed to act as a proxy between INSANEs and the existing building server components, virtually concealing them behind one external interface. Basically, the BSCSM is now used as the sole interface for crowdsourcing access, disallowing direct access to the fingerprinting server and being the only interface with setter-capabilities. The BSCSM recycles the majority of the source code of the INSANE.

A rough overview of the actual current architecture of MapBiquitous is depicted in Figure 7.2.1.

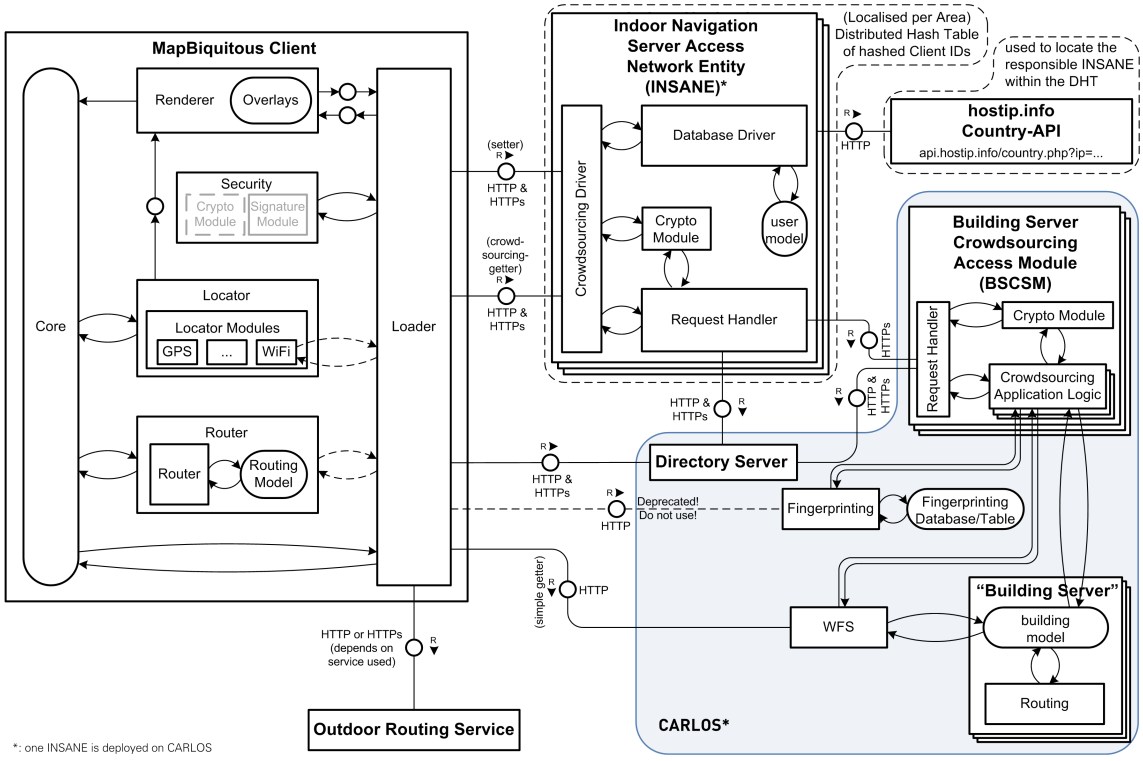


Figure 7.2.1: Actual architecture of MapBiquitous as implemented by 15 October 2012

7.3 THE IMPLEMENTATION

The components newly introduced into the architecture in section 6.2 abstract away from actual implementation details, which – of course – is good, as the implementation itself is proof for

the realisability of the concept. Therefore, a few details of the actual implementation shall be given/discussed here. For reference, the implemented source code is available on the Compact Disc (CD-ROM) accompanying this thesis.

The most import details to point out must definitely be the choice of programming language. All new components, namely the INSANE, the BSCSM as well as the replacement directory server were implemented in PHP. This of course limits deployment of the components to web servers running PHP; however, the entire implementation refrains from system calls, making the components usable on any operating system underlying the web server.

Next, any data to be stored either on an INSANE or BSCSM requires a MySQL database to be available for data storage. This is already true for the original directory server, so this cannot be considered a constraint for the replacement directory server.

Additional constraints emerged during implementation, e.g. the handling of JSON[116] representations of data structures is rather intuitive in PHP when using the `encode_json($construct)`, `decode_json($json)` and `last_json_error()` methods. Unfortunately, `last_json_error()` requires PHP 5.3.0 or newer, which was available on the XAMPP testserver, but not on the actual distribution servers. Therefore, error-handling had to be reduced from 'actual JSON errors' that would have helped to identify errors in the JSON strings to 'valid JSON or not'. But still, `encode_json($construct)`, `decode_json($json)` require PHP 5.2.0 or newer, so the alternative would have been to implemented own JSON handlers. As this would have been to time consuming, the decision was made to set a minimum requirement to the servers later operating as INSANEs and BSCSMs of PHP 5.2.0 or newer. However, in order to not fully give up support of PHP 5.1.* or older, the JSON-handling was outsourced into a `JSON_Handler.php` module[117], allowing easy replacement of the `isJSON($string)`, `fromJSON($json)` and `toJSON($construct)` methods. Lastly, the actual functionality of the encapsulated `encode_json($construct)` and `decode_json($json)` methods is either provided by PHP (PHP 5.2.0 or newer), or the free and open-source `JSON_Library.php`[118]. Also, HTTP-based communication between the INSANE and the BSCSM, the BSCSM and the fingerprinting server as well as the communication between the BSCSM and the geoserver proved to be challenging. Usage of PHP modules such as 'cURL'[119] was to be prevented since it is not foreseeable which PHP modules are available on target systems and which not. Therefore, a socket-based communication using PHP's `fopen` function was implemented in the `HTTP_Requests.php` handler module, requiring no additional PHP modules. The module is closer described in in Programme E.2.1.

Another interesting aspect is the implementation strategy and which effects it had on design decisions as well as server requirements.

First, the actual implementation of the concept conceived in section 6.2 proved to be challenging as the concept could not always be implemented as thought (details were presented in section 7.2), and problems through no fault of one's own postponed implementation ideas, sometimes rendering them unimplementable (details were presented in section 7.1). This also contributed to the decision to choose PHP over Java as the language to implement the INSANE and the BSCSM, especially as quick and dirty prototyping was more accessible to the author via PHP with a XAMPP installation running on localhost. Further, memory consumption[120] was considered in the earliest stages of prototyping. The memory consumption of object-oriented PHP presented itself to be much higher compared to not object-oriented (structured) PHP[121]. This can be explained by the RESTfull request/reply philosophy of the communication at hand:

- Any communication between clients and INSANEs is finalised with one request and one reply, and

[116] JavaScript Object Notation
[117] The module can be found in section E.4.
[119] `http://php.net/manual/en/book.curl.php` – Accessed 4 October 2012
[120] In means of Random Access Memory rather than harddisk space.
[121] A factor of 3 could be measured.

- most of the communication between INSANEs and building servers is finalised with one request and one reply.

Therefore, no demand to keep states, objects, etc. over several requests and/or replies arises, especially as all objects created when handling a request are destroyed after finishing the computation of the reply. Even further, the computation strategy followed is monotonic linear, though it is not strictly monotonic linear as `WHILE`, `FOR` and `FOREACH` loops are used for countable, not infinite repetition of code segments; hence, object-orientation would have only created unnecessary overhead such as object control headers, object reference pointers, etc. Exemplarily, the basic computation flow of a WFS request on a building server is illustrated in Figure 7.3.1. In prevalent code classification, the strategy followed is classified as *imperative structured programming with modularised includes*[122].

Incidentally, modularising the INSANE as well as the building server crowdsourcing module (BSCSM) source code while maintaining 'human readability' of the methods offered required some crafty definitions of scanable syntax. In the end, each method can be automatically loaded into the running INSANE or BSCSM by strictly following the newly created *definition block syntax*, as it is completely parsable[123] by PHP. The structure is rather simple, relying on keywords such as '`DEFINITION BLOCK`' or '`@return`', delimiters such as '`||`' or '`|||`', as well as PHP's own serialisation-syntax, allowing automated usage of PHP's `serialize` and `deserialize` functions. An exemplary method definition block using the defined syntax is presented in section E.5 of the supplementary.

As the observing reader may have noticed, Figure 7.3.1 has forestalled the use of HTTP-based communication. All newly introduced components communicate solely via HTTP and/or HTTPs, exactly as it was conceived in section 6.3. Further, complex data structures are encoded using JSON representations, allowing the transmission of complex data structures within the POST variable headers of the HTTP packets. The constraints emerging from this were discussed earlier.

7.4 CONCLUSION

The proof-of-concept implementation follows the concept conceived for the most part, however deviates in details that could only be uncovered by actually implementing the concept. The directory server was replaced by one allowing lookup of building server as well as indoor navigation access network entities (INSANEs) by geographical position/area and name/identity. All existing building server components were left untouched and virtually concealed by introducing a building server crowdsourcing module (BSCSM) as a proxy interface on the building servers. Both conceived access paths can be found in the implementation, having the crowdsourcing access path span from the client over the INSANE to the BSCSM into the building server, the non-crowdsourcing access path connect the client directly to the getter interfaces[124] of the building server.

PHP was chosen as the implementation language, following the 'imperative structured programming with modularised includes' programming paradigm in order to reduce memory consumption as good as possible. This choice was also enforced by diverse problems that had occured, disallowed modification of existing components and were very time consuming while remaining unsolvable.

[122] The '*modularised includes*'-part originates in the fact that the code is organised into modules which are included on demand.

[123] There are controversial discussions dealing with the existence of the noun 'parsability' and the corresponding adjective 'parsable' (or 'parseable'). However, the author of this thesis assumed the adjective to exist. Should it not exist, the meaning defined by Noam Chomsky shall be valid: 'The ability to assign a structural analysis to sentence'.

[124] E.g. the WFS or the positioning method of the fingerprinting server.

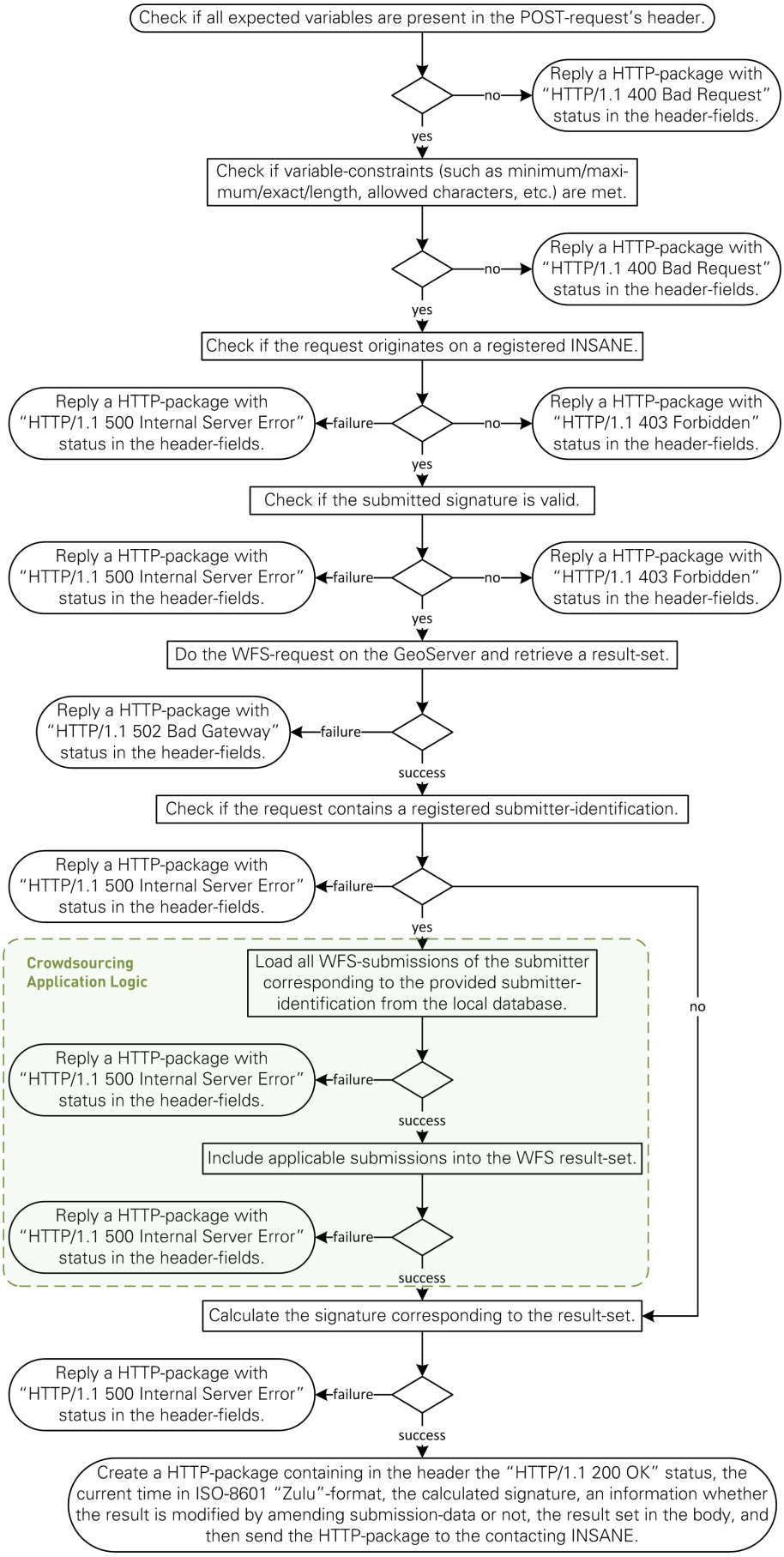


Figure 7.3.1: Application flow of a WFS-request in a building server crowdsourcing module.

Intercommunication with the MapBiquitous clients is guaranteed by having all communication base on HTTP. Larger chunks of data are encoded into JSON-strings, allowing best possible[125] compatibility of the Java-based application logic on the client running under the Android operating system and the PHP-based application logic running on the INSANEs.

[125]Currently that is.

8 EVALUATION

'When I examine myself and my methods of thought, I come to the conclusion that the gift of fantasy has meant more to me than any talent for abstract, positive thinking.'

Albert Einstein

EVALUATION

Before actually starting into this chapter, it shall be noted that the aspects considered within this chapter are limited to the INSANE and building server crowdsourcing module, their inclining system modifications and their communication with each other as well as the clients. Any aspects corresponding to modifications of the client shall not be given herein; the interested reader is requested to kindly refer to [Bom12].

The methods used to evaluate the results can of course only be applied to measurable and comparable aspects of the concept (and implementation) introduced within this thesis. Therefore, the following aspects shall be evaluated:

1. Conformity of the implementation to the design goals.
2. Comparison of old and new communication.
3. Resource use and communication load compared to 'default websites'.
4. Estimation of effort for a real setting compared to the laboratory condition results.

Each aspect will be evaluated in the following sections, beginning with an emphasis on the first two aspects. The third and fourth aspect are evaluated conjointly in section 8.3.

8.1 CONFORMITY OF THE IMPLEMENTATION TO THE DESIGN GOALS

On a superficial level, the design goals defined in section 5.3 are met with the exception of goal 3e, but that is not surprising, as the implementation was focussed on actually complying to all design goals in the best possible manner while managing the implementation in the limited processing time given. Therefore, the deliberate decision not to define and implement encryption was made. Additionally, weekly meetings of the implementing students and their supervisor lead to goal-oriented concept design and implementation. Therefore, the result summarised in Table 8.1.1 are neither surprising nor unexplainable.

Table 8.1.1: Results with respect to the design goals

	Goal	Result	Comment
1	Possibility to not participate in the crowdsourcing.		
1a	The system shall remain usable as to now.	√	Access not related to crowdsourcing remains untouched. (by choice, clients may access the building servers via HTTPs rather than HTTP)
1b	Are modifications inevitable, they shall be as minimal as possible.	√	—
1c	Only basic data shall be accessible.	√	Crowdsourcing data are stored within a separate module; hence, using a non-crowdsourced getter will not allow access to crowdsourced data.
	.		
2	Choice to participate in the crowdsourcing.		

continued on next page ⇝

⇝ continued from previous page

	Goal	Result	Comment
2a	Legal boundaries (e.g. BDSG) are to be adhered.	(√)	In the moment there exists a problem with the HTTPs implementation. ⇝ Refer to goal 3d.
2b	Aware crowdsourcing (A∗C) with crowd awareness shall be supported.	√	Position corrections are supported.
2c	U∗C shall not be obstructed.	√	Different crowdsourcing methods are supported by modularisation of the system.
2d	Participation shall be anonymous against the crowdfunding building servers.	(√)	Users handled by the INSANE set up on the 'CARLOS' server are destined not to be anonymous, as 'CARLOS' acs as building server for all currently available buildings; hence, the administrator of 'CARLOS' can easily deanonymise all users handled by the INSANE running on 'CARLOS'. Only users using the alternative INSANEs set up for testing[126] are currently anonymous, but this is owing the fact that the DHT-distribution of the INSANEs is not functional, yet.
2e	Submissions shall be drerable.	√	With the current database structure, any submission can be deleted, removed, erased and revoked.
2f	Submissions shall only be drerable for a defined time.	(√)	Current implementation drers any submission as soon as it is sifted, otherwise drering is possible without time constraints. However, the submissions are stored with their date of submissions, so basically a time constraint can be implemented using the stored date of submission.
2g	Submissions shall be reviewable.	(√)	Database structure is prepared for reviewing, but reviewing is not implemented.
2h	Submissions shall be siftable.	(√)	Database structure is prepared for sifting, but sifting is not implemented.
	.		
3	The crowdsourced system architecture must be extensible and secure.		

continued on next page ⇝

[126] They are reachable at `http://insane.the-tester.de/` and `http://insane.hara.tc/`

↝ continued from previous page

	Goal	Result	Comment
3a	Required application logic shall be easily replaceable.	(√)	Application logic can be modified, removed or added by simply loading a new module; however, the module requires a defined syntax, which is explained, but requires understanding, otherwise the module or even the entire infrastructure component stops working.
3b	The source code of new components shall be self explanatory and easily extensible rather than highly optimised.	(√)	Some optimisations were inevitable as the system had to be made of good performance in order the acquire scalability. Additionally, some code fragments considered basic knowledge to programmers were left uncommented.
3c	Data ascertainment must be extensible and configurable.	(√)	Data to be ascertained and the ascertainment itself can be modified, removed or added by defining corresponding rules and fragments in the application logic. Therefore, the same constraints as for goal 3a apply.
3d	Communication must be encrypted.	(√)	Basically, encrypted communication is prepared and usable, but due to some incomprehensible problems with the Android implementation of HTTPs and certificate handling in PHP, encryption is currently deactivated via a configuration setting.
3e	Data stored on servers must be encrypted.	×	Currently, only passwords are 'encrypted' by only storing their SHA-256 hash. All other data is stored in plain!
3f	User shall be bound to infrastructure components only loosely.	√	Users are free in their choice of either contacting building servers directly or via an INSANE. Further, the corresponding user profiles are stored on the INSANEs and only amended by ACL entries on the building servers. Further, the INSANE to contact is non-rigid as the DHT-based distribution envisages separate hash-tables for different regions; hence, a user would contact different INSANEs in different regions of the world, while having a guarantee of their data being accessible.

continued on next page ↝

⇝ continued from previous page

	Goal	Result	Comment
3g	GSM data shall be ascertained and stored for future processing and information extraction.	√	GSM fingerprints can be created and are stored in the crowdsourcing module. However, the stored data is not correlated to other building data, yet. Therefore, making use of the stored data is possible in the future.
3h	As proof of system function, at least one crowdsourcing method must be functional.	√	Position corrections are supported.

The table assumes availability of the results of [Bom12].

8.2 COMPARISON OF OLD AND NEW COMMUNICATION

An interesting aspect of evaluation definitively is the communication overhead generated by newly introduced modules, etc. However, overhead can only be added to existing communication, so basically only the non-crowdsourcing access of clients on building servers can be compared to the anonymised crowdsourcing getter-access proxied via an INSANE. However, another communication can be evaluated, even though this should not be possible: creation of WLAN fingerprints, which is – as a reminder – a crowdsourcing-setter. This lucky fact originates in the non compliant implementation of the fingerprinting server in [Gru12], where it was introduced as a means to enhance positioning quality, but it was not explicitly considered crowdsourcing.

Before amending the MapBiquitous architecture with crowdsourcing capabilities, the retrieval of WFS data as well as (WLAN) fingerprinting-based positions was already present. The WFS data were requested from the building servers, the fingerprinting-based positions from the fingerprinting server. Both are still possible, but the access to the fingerprinting server is considered deprecated and shall not be used. In the future, direct access to the fingerprinting server will most likely be blocked on `.htaccess`-level.

8.2.1 Comparison of the WFS requests

First, focussing on the WFS requests a clear overhead can be recognised, which shall be exemplarily discussed for the WFS request `http://carlos.inf.tu-dresden.de:8080/geoserver/tud_inf/wfs?SERVICE=wfs&VERSION=1.0.0&REQUEST=GetFeature&TYPENAME=tud_inf:TUD_INF_G&`. This is a simple GET request, returning a XML-document with a total size of 2088 Byte, of course considered without HTTP headers, TCP headers, etc. This negligence of the headers is important, as their actual size does not matter, since the evaluation shall show the additional overhead to be added to the original data. The same applies for the original request, in which only the actual data are considered: For the GET request the actual data are the GET variables (`?SERVICE=wfs&VERSION=1.0.0&REQUEST=GetFeature&TYPENAME=tud_inf:TUD_INF_G&`) with a total size of 72 Byte. In sum, the total size of the communicated data (request and reply) is 2160 Byte for the non-crowdsourcing direct access of the client on the building server.

Now, aiming at the same data to be exchanged, the crowdsourcing getter-access proxied of an INSANE shall be evaluated. This evaluation is split into two parts for reasons of comparability.

The first part evaluates the communication between client and INSANE (encapsulated in the columns of Table 8.2.1 headed 'CS communication'), while the second part evaluates the communication between INSANE and building server (encapsulated in the columns of Table 8.2.1 headed 'BS communication').

Assuming the user already has a valid account registered on the INSANE they are contacting[127], the results given in Table 8.2.1 and Table 8.2.2 emerge for different requests, all following the same size measuring conventions as described earlier. The sizes given in parenthesis are the sizes including hash-based signatures[128]. Further, the column headed 'Overhead' represents the overhead over the entire communication-path, not limiting this aspect to the packet level.

Table 8.2.1: Results for '`getFeature`' on 'tud_inf:TUD_INF'

	Non-CS communication		**CS communication**		**BS communication**		
TYPENAME	**Request**	**Reply**	**Request**	**Reply**	**Request**	**Reply**	**Overhead**
`tud_inf:TUD_INF_G`	72 Byte	2088 Byte	206 Byte (782 Byte)	2118 Byte (2668 Byte)	159 Byte (739 Byte)	2118 Byte (2668 Byte)	2441 Byte (4697 Byte)
`tud_inf:TUD_INF_E0`	73 Byte	84060 Byte	207 Byte (787 Byte)	84090 Byte (84670 Byte)	160 Byte (740 Byte)	84090 Byte (84670 Byte)	84414 Byte (86734 Byte)
`tud_inf:TUD_INF_E1`	73 Byte	118849 Byte	207 Byte (787 Byte)	118879 Byte (119429 Byte)	160 Byte (740 Byte)	118879 Byte (119429 Byte)	119203 Byte (121463 Byte)
`tud_inf:TUD_INF_E2`	73 Byte	123816 Byte	207 Byte (787 Byte)	123846 Byte (124426 Byte)	160 Byte (740 Byte)	123846 Byte (124426 Byte)	124170 Byte (126490 Byte)
`tud_inf:TUD_INF_E3`	73 Byte	127849 Byte	207 Byte (787 Byte)	127879 Byte (128459 Byte)	160 Byte (740 Byte)	127879 Byte (128459 Byte)	128203 Byte (130523 Byte)

Results are based on actual communication using the INSANE at `insane.the-tester.de`.

Table 8.2.2: Results for '`getFeature`' on 'tud_inf:TUD_INF' in %

TYPENAME	**Overhead Client↔INSANE**	**Total Overhead**
`tud_inf:TUD_INF_G`	7.59 (59.72)	113.01 (117.45)
`tud_inf:TUD_INF_E0`	0.19 (1.57)	100.33 (103.09)
`tud_inf:TUD_INF_E1`	0.14 (1.09)	100.24 (102.14)
`tud_inf:TUD_INF_E2`	0.13 (1.07)	100.23 (102.10)
`tud_inf:TUD_INF_E3`	0.13 (1.04)	100.22 (102.03)

Results are based on actual communication using the INSANE at `insane.the-tester.de`.

[127] Should there be no such account, an unforeseeable overhead is created by – in the worst case – having the user contact several INSANEs until they have reached the responsible INSANE in their geographical region and have completed registration.

[128] The signature is calculated for the hash of the data rather than the data itself in order to save on computing effort.

The results seem unexpected, but they are actually very logical. All communication needs to be signed and amended with additional data such as the client-ID, the user's password, a time-stamp, etc. Therefore, by definition of the interfaces used, the measurable overhead is created[129]. However, the numerical overhead is constant, as the same data are amended, and the chosen signing algorithm always generates signatures of the same length. In summary for the example of `getFeature`-requests on the building server 'tud_inf:TUD_INF', communication proxied via an INSANE always adds 130 Byte for the request from client to INSANE, another 160 Byte for the communication from INSANE to the building server, as well as 30 Byte for the reply from the INSANE to the client and the original data size plus 30 Byte for the reply from building server to the INSANE. With the hash-based encryption, all four communications are amended with 580 Byte signature data each, ergo 2320 Byte total signature data.

As data amendments are constant and only the size of the original data varies, the variable overhead lies in the result-packet transmitted from the building server to the INSANE. However, the variable quantitative overhead only varies in actual data size, the qualitative overhead percentage generated by the data is at a constant 100%, and the variable qualitative overhead is strongly correlated with the constant size on the amendments. Therefore, the quantitative overhead is strongly correlated linearly and the qualitative overhead is reverse polynomial correlated to the size of the original data, as displayed in Figure 8.2.3. The results lead to the following conclusions:

$$\lim_{\text{data size}\to\infty} \text{Overhead}_{\text{Client}\leftrightarrow\text{INSANE}} = 0\% \tag{8.2.1}$$

$$\lim_{\text{data size}\to\infty} \text{Overhead}_{\text{total}} = 100\% \tag{8.2.2}$$

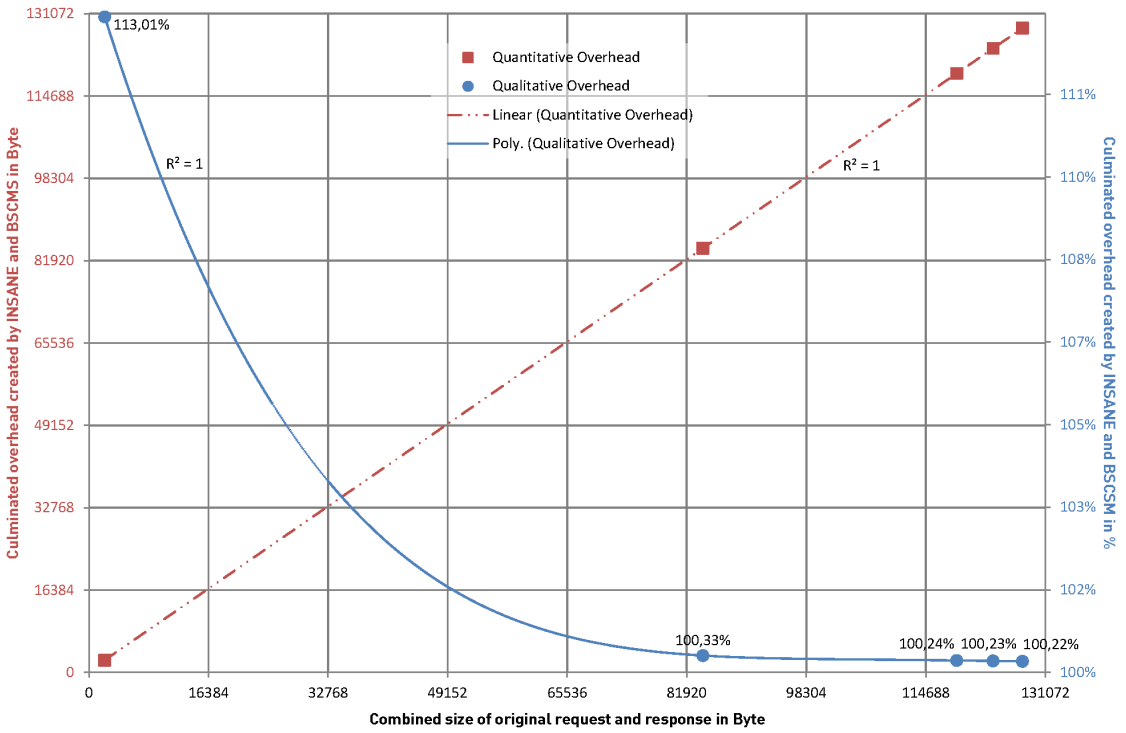


Figure 8.2.3: The quantitative and qualitative WFS overheads

[129]For details on the amended data defined by the interfaces, refer to Appendix H.

Even though expected, considering the fact that the INSANEs proxy all communication, the result in Equation 8.2.2 is outstanding, as the overhead only results in a duplication of the transmitted data without any further overhead. – Considering the possibility of compression[130], there is clear potential for reduction of the quantitative overhead.

The result in Equation 8.2.1 does not surprise, as the total overhead is a constant 160 Byte (1320 Byte with encryption), the overhead can be neglected for the data to be expected for WFS requests. – Once again, considering the possibility of compression[130], there is clear potential for overhead reduction.

However, both results need to be considered with care, as both base on measurements of unencrypted HTTP traffic. Encrypted HTTPs communication adds an additional overhead, but as this has been discussed in a variety of academic as well as general sources[131], it shall not be discussed here.

Summarising, the results are as to be expected for standard web servers and proxy servers handling resources. In this spirit, the crowdsourced MapBiquitous architecture[132] neither worsens nor enhances the amount of data to be transmitted between client and server[133].

8.2.2 Comparison of WLAN-Fingerprinting Communication

Other than the WFS requests, there is no conceptual difference between the two types of fingerprinting. As described in section 7.2, the newly created crowdsourced WLAN fingerprinting actually uses the same fingerprinting service and interacts with the same interfaces, as the old MapBiquitous did before crowdsourcing was introduced. Therefore, this comparison might even generate better conceivable results than the previous subsection.

There are two cases to be considered, first uploading a fingerprint ('fingerprinting') and second retrieving a position based on a fingerprint ('positioning').

Fingerprinting: Uploading a WLAN-Fingerprint

For the first case, the default configuration used by MapBiquitous requires ten samples to be uploaded for a fingerprint. The upload itself has three phases[134]:

1. • Send a HTTP GET request with a random integer to '`start?id=RNDINT`'.
   The only requirement towards the random integer is for it to be in-between 0 and $2^{32} - 1 = 4294967295$. As this random integer is transmitted as a GET variable, it has a minimum transmission size of 1 Byte (< 10) and a maximum transmission size of 10 Byte (> 999999999).
   As the possibility of either of the integers within the interval is equal, the average transmission size of the request in phase one is 9 Byte ('`start?id=`') plus 9.7413 Byte[135] (the random integer); hence a total of 18.74 Byte for the GET variables.

[130] Most web servers support GZIP compression.
[131] In fact, so many that this can be considered general knowledge to computer scientists.
[132] With neither SSL encryption nor GZIP compression.
[133] As the INSANE acts as proxy for building server access, it can be considered a server.
[134] Exemplary for the fingerprinting server at `http://carlos.inf.tu-dresden.de:8080/fingerprints/clustered/`
[135] Average size: $\frac{(10 \cdot 1 + 90 \cdot 2 + 900 \cdot 3 + 9000 \cdot 4 + 90000 \cdot 5 + 900000 \cdot 6 + 9000000 \cdot 7 + 90000000 \cdot 8 + 900000000 \cdot 9 + 3294967295 \cdot 10)\,\text{Byte}}{4294967295} = 9.7413\,\text{Byte}$

- Add the position at which the fingerprint was generated and number of samples as POST variables to the GET request[136].
  The POST variables look something like
  `position={"building":"tud_inf:TUD_INF","latitude":51.02567595421748, "wt_sec":0.0,"longitude":13.723133764723404,"heading":0.0,"altitude":0.0, "timestamp":1349963850,"accuracy":0.0,"level":3}&nr_of_samples=7`
  As the size of the longitude and latitude vary[137], this evaluation shall focus on the faculty building of the Faculty of Computer Science at TUD. Both latitude as well as longitude may vary in the 14th and 15th decimal place. Further, the amount of samples may vary depending on how many WLAN access points are visible to the client-device at the location the fingerprint is created. As more than nine access points can very seldom be seen at the same time, it shall be assumed that the size of the sample count is constant at 1 Byte, especially yielding an average visibility of 6 access points. Hence, the average size of the POST data is 205 Byte.
- **Total average size of GET and POST data in phase one:** 223.74 **Byte**

2.
- Send a HTTP GET request using the same integer as in the first phase to '`sample ?id=RNDINT`' for each sample of the fingerprint.
  As in phase one, the random integer adds 9.7413 Byte to the size of the GET data plus 10 Byte for the variable and path; hence a total of 19.74 Byte for the GET variables.
- Add the sample data as POST variables to the GET request[136].
  The POST variables look something like
  `sample={"ap_mac":"08:17:35:33:59:00","y":0.0,"x":0.0,"latitude":0.0, "longitude":0.0,"ap_signal":-68,"id":0,"floor":0}`
  As the signal-strength of the access points is measured in integer decibel values, the `ap_signal`'s size can be assumed to be constantly 3 Byte; therefore, the POST data generate a transmission size of 117 Byte for each sample.
- **Total average size of GET and POST data in phase two:** 136.74 **Byte**

3.
- Send a HTTP GET request using the same integer as in the first and second phase to '`finish?id=RNDINT`'.
  Added to the 11 Byte for the variable and path, the random integer adds 9.7413 Byte to the size of the GET data plus, same as it did in phases one and two; hence a total of 20.74 Byte for the GET variables.
- As there are no POST data, the third phase does not have any data size for the POST data.
- **Total average size of GET and POST data in phase three:** 20.74 **Byte**

Of course, each packet transmitted to the fingerprinting server results in a '`HTTP/1.1 200 OK`'-reply, but the reply has no body; therefore, the reply shall be neglected. The average total size of the data transmitted can be assumed to be $(223.74 + n \cdot 136.74 + 20.74)$ Byte, with n being the amount of samples to be transmitted.

In the correct implementation treating WLAN fingerprinting as a variant of crowdsourcing, clients must transmit their submissions to an INSANE from where they are forwarded to the corresponding building server. Even though implementation reality still uses the dedicated fingerprinting server, it shall be assumed each building server has its own fingerprinting service.

The major difference between the old and deprecated fingerprinting interface and the new crowdsourced interface is the reduction from eight (in average) requests to one. Basically, the schema follows the concept investigated earlier for the WFS request. The proxy operation culminates in a request from client to the INSANE of a length of $(353 + n \cdot 117 + n)$ Byte[138] [129], in which the $(n \cdot 117 + n + 1)$ Byte added to the 352 Byte base originate in the previously mentioned 117 Byte

[136] This faulty implementation violating the HTTP standard was given by the existing fingerprinting server.
[137] Up to 15 decimal places are supported, so the size of the latitude can range between 3 Byte (0.0) and 19 Byte (–89.999999999999999), and the size of the longitude can range between 3 Byte (0.0) and 20 Byte (–179.999999999999999).
[138] The size yields an assumption of a password of length 8 and the building server 'tud_inf:TUD_INF' being contacted.

per sample, $n - 1$ commas separating the samples from each other as well as 2 Byte for the array-parenthesis [and] of the encapsulation JSON-String. Using hash-based signatures[128], an additional 580 Byte must be accounted. As true for the deprecated interface, the size of the reply shall be neglectable.

Following the transmission from client to the INSANE, the fingerprint data must be forwarded to the corresponding building server. This culminates in additional $(464 + n \cdot 117 + n)$ Byte[139] [129] of data to be transmitted from the INSANE to the building server, now having the previously explained $(n \cdot 117 + n + 1)$ Byte added to the 463 Byte base. Once again, using hash-based signature[128] adds another 580 Byte.

The previously calculated results are summarized in Table F.1.1 in the supplementary; once again, the number in parenthesis represent the results with encryption. Further, the results of Table F.1.1 are summarised into Figure 8.2.4 as well as:

$$\lim_{\text{samples} \to \infty} \text{Overhead}_{\text{Client} \leftrightarrow \text{INSANE}} = -15.56\% \tag{8.2.3}$$

$$\lim_{\text{samples} \to \infty} \text{Overhead}_{\text{total}} = 68.89\% \tag{8.2.4}$$

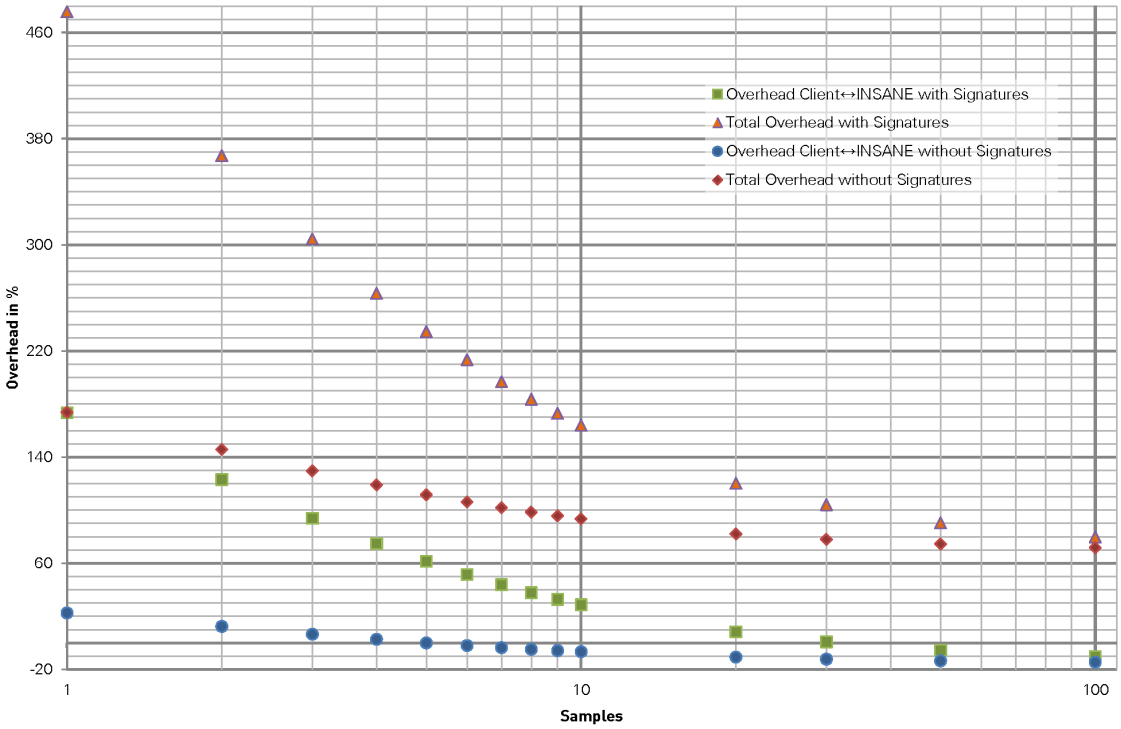


Figure 8.2.4: The percentual overheads for WLAN fingerprinting

The results seem rather surprising. No compression[130] is considered, but the overhead is negative for the client↔INSANE communication and abundantly clear below 100% for the entire proxy communication. However, these results are only valid for many samples to be uploaded, i.e. 100 or more. Considering the default setting described earlier, only an average of 6 WLAN access points are visible at a position to be fingerprinted in the faculty building; hence, numbers to be considered 'practical' range between –3.57% and 2.68% for the client↔INSANE communication (43.87% and 74.87% respectively for signed[128] communication), and in a range from 101.85% to 119.06% (196.37% to 263.44%) for the total proxy communication.

[139] The size yields an assumption of the INSANE handling the request being 'insane.the-tester.de' and the size of the model-description being 20 Byte.

Looking at the revised (i.e. realistic) results, the same conclusion can be drawn as for the WFS requests: The crowdsourced MapBiquitous architecture neither worsens nor enhances the amount of data to be transmitted between client and server. However, as soon as the packets are signed[128], the average overhead to be expected is rather unacceptable, clearly showing a demand to utilise compression[130] for packet transmission.

Positioning: Determining the Position based on a WLAN-Fingerprint

For the second case, the default configuration used by MapBiquitous actually only requires the client to transmit a few access point data (generated by the same method that is used for the fingerprinting) to the fingerprinting server. Therefore, the size of the transmitted data is strongly correlated to the amount of access points visible at the location of the client, as well as the amount the client is willing to transmit. To ease the evaluation, it shall be assumed that all visible access points are considered for positioning and hence their information needs to be transmitted to the server. Differing from fingerprinting, each access point sample has an average size of 50 Byte. Added to this data size are $n + 2$ Byte for the commas separation the n samples as well as for the array-parenthesis [and] of the encapsulation JSON-String. Further, the size of the actual request-path, which is 8 Byte for '`position`', is added. as well as the size of the data of the previous position, which is used as a calculation base by the fingerprinting server; it is averaged at 200 Byte. As these data are transmitted as POST data, the corresponding POST variable identifiers (`accesspoint`= and `&prev_position`=) must be accounted for by adding another 28 Byte to the data size. – Summarising, the deprecated positioning interface requires the client to transmit $(238 + n \cdot 50 + n)$ Byte.

The newly created crowdsourced interface[129] which is only accessible via an INSANE only adds a constant to the data size, which in the course of this evaluation is 164 Byte[140] (744 Byte with signature[128]), resulting in a total data size of $402 + n \cdot 50 + n$ Byte ($982 + n \cdot 50 + n$ Byte) for the client↔INSANE communication.

The forward to the corresponding building server yields the transmission of $424+n\cdot50+n$ Byte[141] from the INSANE to the building server. As before, signing[128] the packet results in additional 580 Byte to be transmitted.

Differing from the fingerprinting, positioning yields a server-side reply, namely the position calculated by the server. This reply has an average size of 200 Byte (780 Byte with signature). It has to be relayed via the INSANE back to the client.

The calculations for different sample amounts are summarised into Table F.2.1. Further, the result of Table F.2.1 are visualised in Figure 8.2.5 and show a clear trend:

$$\lim_{\text{samples}\to\infty} \text{Overhead}_{\text{Client}\leftrightarrow\text{INSANE}} = 0\% \tag{8.2.5}$$

$$\lim_{\text{samples}\to\infty} \text{Overhead}_{\text{total}} = 100\% \tag{8.2.6}$$

The results given in Equation 8.2.5 and Equation 8.2.6 seems outstandingly good, as they suggest that only the expected duplication of the transmitted data due to the proxy communication occurs. However, as true for the fingerprinting results, these results delude, as they are only valid for large amounts of samples. Once again, assuming the average of 6 WLAN access points visible in the faculty building, more realistic numbers emerge from 23.53% to 41.94% for the client↔INSANE communication (106.74% to 190.28% for signed[128] communication) as well as from 150.22% to 189.51% (316.64% to 486.19%) for the entire communication path.

[140]Assuming a pathword-length of 8 Byte and the building server contacted being 'tud_inf:TUD_INF'.
[141]The size yields an assumption of the INSANE handling the request being 'insane.the-tester.de'.

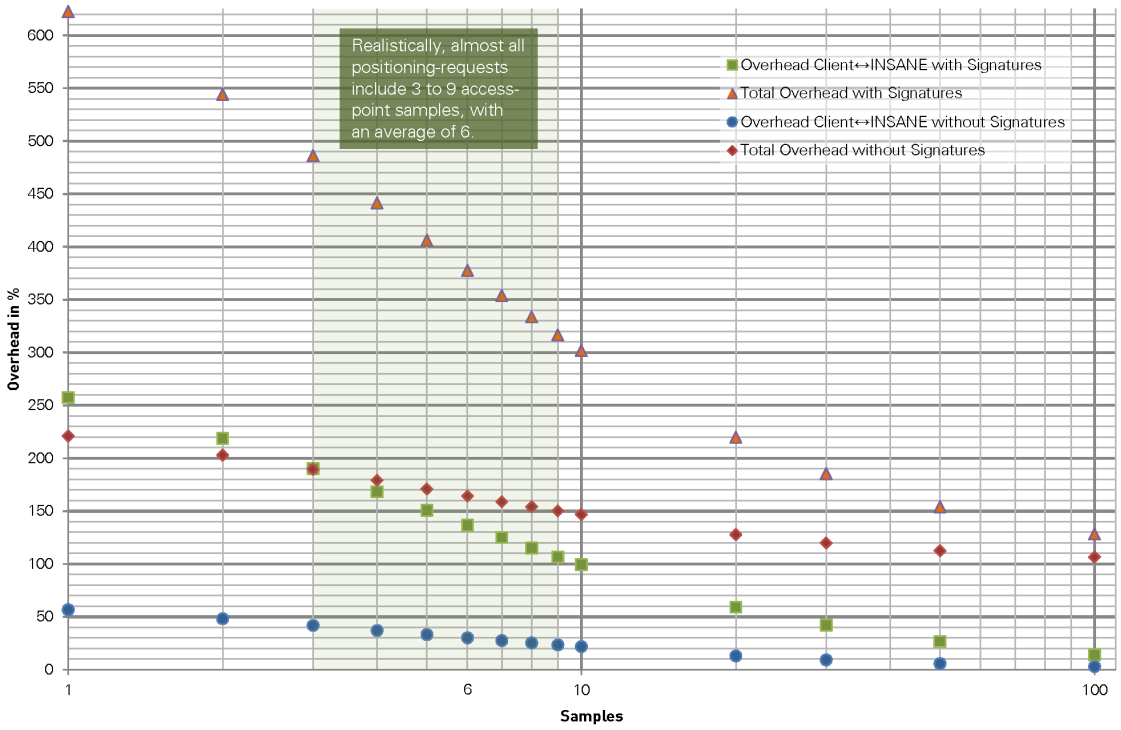


Figure 8.2.5: The percentual overheads for WLAN positioning

Especially the revised results for signed communication are totally inacceptable with an expected average overhead of over 377.57% for the entire communication path, being almost the quadruple of what is to be expected for a communication via one intermediate proxy. – Utilisation of compression[130] must definitely be considered for the communication between the INSANEs and the building servers in order to reduce the quantitative overhead.

The expectable overhead of below 25% when considering only the client↔INSANE communication can be deemed 'still acceptable'; however, as soon as packets are signed[128], the overhead rises up to nearly 100%, which is unacceptable. – Once again, utilisation of compression must definitely be considered.

8.2.3 Conclusion for Fingerprinting and Positioning

The evaluation results seem unacceptable. However, one should not forget that the original communication before crowdsourcing was introduced did not pay any respect to security. Anybody could access the interface and create fingerprints. Now, after crowdsourcing has been introduced and the fingerprinting server should only be reachable via an INSANE, passwords and valid user accounts are required. Taking into consideration that the original packet sizes were quantitatively small, the overhead created by the passwords, client-IDs, etc. surely boosts the qualitative overhead into enormous levels. With respect to the now available security due to authentication, this qualitatively high overhead is a fair price to pay.

8.3 RESOURCE USE AND COMMUNICATION LOAD COMPARED TO 'DEFAULT WEBSITES'

As it is hardly possible to measure scalability with hard numbers since physical measurands are strongly correlated to the hardware used, the connection speed, et cetera, a comparative approach of evaluation seems to be the method of choice. Therefore, it shall be assumed that the INSANE as well as the BSCSM can be considered websites offered by web servers; then a direct comparison to websites operating on the same or very similar hardware can be conducted and in terms of propagation the scalability can be estimated.

In order to have a ground for the comparison, the basic hardware facts of the evaluated setup are given in Table F.3.1 as well as Table F.3.2 (both in the supplementary). Originally, four settings were used for the series of measurements; however, all four series uncovered some incomprehensible server and/or network problems in the faculty building of the Faculty of Computer Science at TUD, as they all shared the server 'CARLOS' as provider for building server functionality. Therefore, a fifth setting was used, relying only on 'external' resources by using three VPS-based setups, one for the INSANE, one for the BSCSM and one simulating a majority – i.e. 85.4% – of the clients. However, the first four settings are not lost and their results actually may help to improve the quality of the network in the affected parts of the faculty building; therefore, they are provided in Appendix I.

For the actually considered setting, the following determining factors applied:

1. The machines simulating the clients were placed in an apartment[142] as well as a server farm north-west of Dresden, Saxony.
2. An actual INSANE at `http://insane.the-tester.de` was used.
3. The VPS hosting the INSANE at `http://insane.the-tester.de` as reconfigured to provide as a dummy-BSCSM. I.e., the VPS was set up to run the actual building server crowdsourcing module application logic, but any actual building server processing[143] was simulated by delaying the reply randomly from 20ms to 70ms.

As for the four unaccounted settings, the basic idea for this setting was to start into the evaluation of resource use and communication load by proving that the results are strongly correlated to the hardware and general load of the servers. This would prove comparability of the scalability results to general web servers, and as such would allow to simply apply findings of web server tests to the MapBiquitous server components.

For the test setting, the interfaces descriptions as given in Appendix H applied[144]. System loads of the INSANE as well as the BSCSM were directly recorded from '`uptime`', and the network statistics were retrieved via 'Wireshark'.[145]

Measuring series for the setting were conducted for different amounts of parallel client requests and were repeated ten times. The results were then averaged into one representative table.

The results for the setting can be found in Table F.4.1. Additionally, they are visualised in Figure 8.3.1.

The results clearly show a strictly linear correlation between the amount of parallel client requests and the packet losses as well as the successful replies. This is not surprising as the communication follows a linear path, from the client to the INSANE, to the BSCSM, back to the

[142] Connected to the internet via a DSL-6000 connection.
[143] Like calling the WFS, generating a fingerprint, etc.
[144] As a reminder: The deployed BSCSM did not do any actual building server related processing.
[145] Wireshark 1.8.3 was used in recording mode. The log-files were evaluated after the measurements, as live-evaluation would have falsified the results by adding further load to the machines.

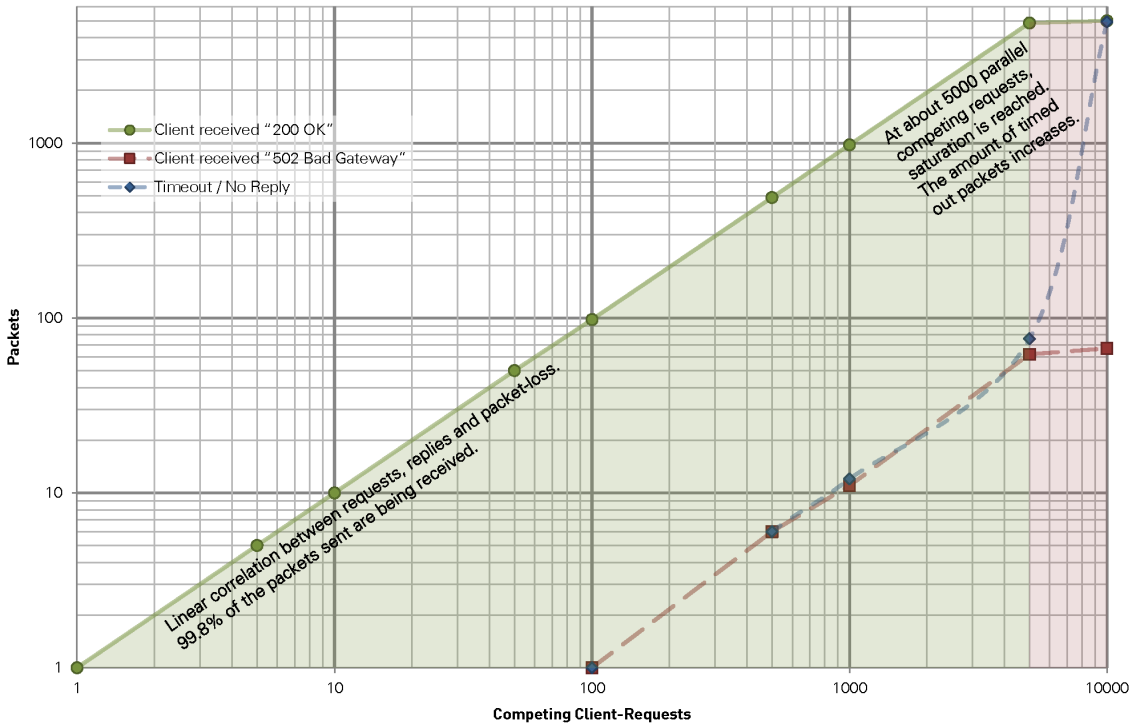


Figure 8.3.1: Packet transmission statistics for the performance and scalability test

INSANE and finally back to the client. Merely the saturation at about 5000 parallel competing requests breaks the linear correlation; however, this is expected, as the underlying Apache httpd web servers of the INSANE as well as the BSCSM do normally fail to process all requests at that high degree of parallelism. Therefore, the deployed INSANE and BSCSM behave as would be expected from any 'normal' website hosted on the same VPS.

Summarising, the INSANE and the BSCSM as implemented both follow the operating figures of any general web server. They neither improve nor worsen the performance and scalability of the underlying Apache httpd web server. Therefore, the numerous recommendations that are given for web servers all over the web[146] can be applied without further refinement to the INSANE as well as BSCSM.

8.3.1 Exemplary Recommendations for Deployment

As previously mentioned, there are in fact so many recommendations on how to optimise performance of a web server, that this information can be considered general knowledge – at least for computer scientists. Therefore, rather than conceiving an own recommendation for an actual deployment, the author of this thesis will pick an exemplary recommendation available in the internet. As it exhaustively explains a lot of what it actually recommends with background information and corresponding calculations, et cetera, the author of this thesis wishes to refer to the book 'Practical mod_perl' by Stas Bekman and Eric Cholet[147].

In chapter 11 of 'Practical mod_perl' the memory consumption depending on different directives of the Apache httpd web server is analysed. This of course having `mod_perl` activated. As the

[146] E.g. just google 'Performance Tweak Apache'.

[147] It is published under the ISBN `0-596-00227-0` and can be ordered with the order number '2270' on `http://modperlbook.org/`. As it is licensed under a Creative Commons Attribution Share Alike license (CC-BY-SA), referencing it within this thesis is legal und justifiable.

previously used evaluation setting had `mod_perl` activated on all three VPS, the recommendations from 'Practical mod_perl' can be applied without further refinement.

Of course, the setting used in the evaluation is far from actual server usage to be expected from the MapBiquitous project in the near future. This is owing to the fact that the server configurations used in the setting are optimised to run a social community with burst access loads at certain hours of the day[148]. For MapBiquitous, it is fair to assume that the near future will not see more than 20 users accessing an INSANE at the same time. However, the setup is scalable and supports more competing accesses by simply adjusting the directives correspondingly.

For the identified demand to support 20 parallel user accesses, the following recommendations shall be given based on the system specification of the machine 'CARLOS' (refer to Table F.3.1 in the supplementary) and the information provided in 'Practical mod_perl':

- **Reserve 256 MB RAM for Apache httpd.**
  Of the 2 GB of available RAM on 'CARLOS', 256 MB shall be dedicated to Apache httpd and `mod_perl` together.
- **Do not modify `HARD_SERVER_LIMIT`.**
  `HARD_SERVER_LIMIT` does not need to be edited as a `MaxClients`-value of 250 is sufficient. This also allows use of Apache httpd 'out of the box', as it does not need to be recompiled.
- **Set `MaxClients` to 25 iff memory-sharing is disabled.**
  Without memory-sharing enabled, the expectable size of each httpd child process is 10 MB; therefore, the available RAM (reminder: 256 MB) leads to a `MaxClient`-value of 25, which is sufficient to server the expected 20 parallel user accesses while maintaining a reserve of 20%.
- **Set `MaxClients` to 62 iff memory-sharing is enabled.**
  With memory-sharing enabled and assuming having 6 MB of shared memory per child process, the `MaxClient`-value can be set to 62, yielding a reserve of 210% over the expected 20 parallel user accesses.
- **Do not deactivate `KeepAlive`.**
  For the expected access behaviour of clients towards INSANEs and INSANEs towards BSCSMs, it is recommendable to keep the `KeepAlive`-directive enabled, ensuring that TCP connection keep alive is used with the HTTP 1.1 connection. It significantly reduces the connection establishment and termination overhead, which automatically reduces the server-load. Additionally, the desired SSL-encryption of the communication (reminder: HTTPs shall be used rather than HTTP) benefits from keep-alive connections as the key-exchange, etc. generate several packets of overhead and it would be very ill-advised to establish, use and terminate a separate TCP connection for each packet.
- **Leave `MaxKeepAliveRequest` at the default of 100.**
  The default `MaxKeepAliveRequests` values of 100 is sufficient, as this allows an average of 5 keep-alive requests per expected user access.

It should be obvious that the speed of the CPU only has an impact on the response time. Having all user access originate on mobile devices such as smartphones, there is no demand for shortest possible response times; hence 'CARLOS' is currently overpowered with its 2.4 GHz CPU.

These recommendations allow easy adaptation to increasing parallel user access behaviour. Having memory-sharing enabled, these recommendations allow up to 49 parallel user accesses without any modifications while maintaining the recommended reserve of 20%. However, should more than 49 parallel user accesses be expected, it is sufficient to simply increase the size of the RAM dedicated to Apache httpd and `mod_perl`. Maintaining the 6 MB of shared memory per child, the following scalability formula emerges:

[148] `http://www.hey-ai.com/` has had over 11000 users with about 250 accessing at the same time in peak periods, generating round about 4750 competing requests.

$$
\begin{aligned}
\text{Dedicated RAM required} &= ((\text{child memory}) - (\text{shared memory per child})) \cdot (\texttt{MaxClients}) \\
&\quad + (\text{shared memory per child}) \\
&= (10\,\text{MB} - 6\,\text{MB}) \cdot (\texttt{MaxClients}) + 6\,\text{MB} \\
&= (4 \cdot \texttt{MaxClients} + 6)\ \text{MB} \qquad (8.3.1)
\end{aligned}
$$

Applying Equation 8.3.1 to different amounts of parallel user accesses, the results of Table 8.3.2 emerge, which are also visualised in Figure 8.3.3. It is obvious that the correlation is strictly linear, which is exactly the expected result.

Table 8.3.2: RAM recommendations with respect to scalability

Parallel User Accesses	**With** 20% **Reserve**	**Dedicated RAM required**
49	59	242 MB
50	60	246 MB
60	72	294 MB
70	84	342 MB
80	96	390 MB
90	108	438 MB
100	120	486 MB
200	240	966 MB
300	360	1.4 GB
400	480	1.9 GB
500	600	2.4 GB

Results are based on Equation 8.3.1.

8.4 CONCLUSION

As the results of the evaluation clearly show, the performance of the newly introduced components depends on the hardware they are deployed on and the speed of their communication lines. This conclusion can be proven by the striking similarity of the newly implemented INSANE and BSCSM deployed in VPS-environments to the performance of prevalent web servers. Additionally, as the INSANE and the BSCSM share the same core application logic and the strategy of their design is the same, namely *imperative structured programming with modularised includes*, the assumption that the same conclusion applies seems fair and justified.

With respect to scalability, the evaluation clearly shows that the same basic constraints apply as for any regular web server. Therefore, any given scalability result for different hardware can be applied to the INSANE and BSCSM, as well. The newly implemented components neither enhance nor worsen the performance of the underlying web server[149].

[149]As a reminder, the underlying web-serer is *Apache httpd* with PHP.

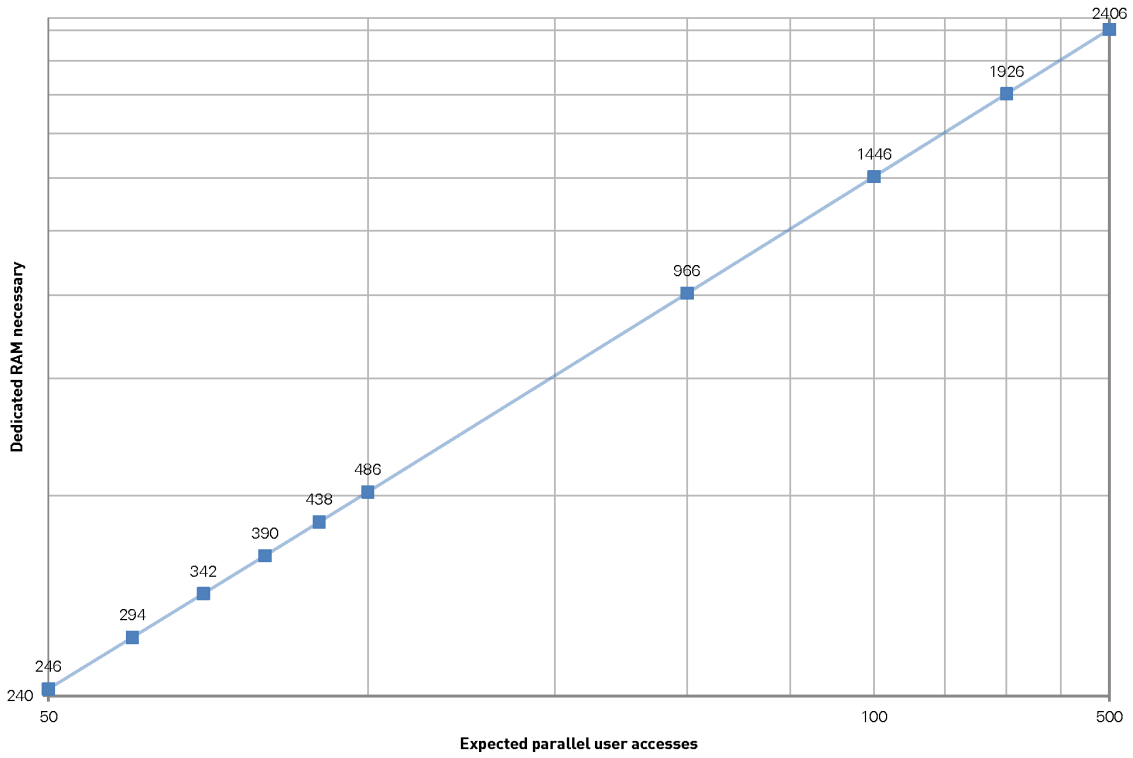


Figure 8.3.3: RAM recommendations with respect to scalability

9 CONCLUSION AND FUTURE WORK

'It's more fun to arrive a conclusion than to justify it.'

Malcolm Forbes

CONCLUSION AND FUTURE WORK

9.1 CONCLUSION

As the concept introduced in chapter 6 has collided with a vast variety of problems as shown in chapter 7, before actually considering the 'real' results and conclusions of this thesis, a first clear conclusion can be drawn: The task of conceiving, implementing and proving an automated means for implicit crowdsourcing could not be pervaded. Rather, the need for a reliable architecture to actually distribute and process crowdsourcing – independent of it being implicit or explicit – was identified. As such an architecture did not exist, the focus of the practical portion of this thesis shifted from actually implementing implicit crowdsourcing to preparing MapBiquitous for crowdsourcing in general. Finally, the team effort with Gerd Bombach totally postponed automated processing of implicitly crowdsourced data. Instead, the above mentioned reliable architecture was conceived and implemented for crowdsourcing in general, having the explicit position corrections introduced in [Bom12] run as an application proving feasibility for explicit crowdsourcing on the newly created architecture. Therefore, the implicit crowdsourcing aspect was reduced to only collecting (GSM) data *implicitly*, without actually processing the data and extracting information on the building servers. Would the processing time of this thesis not have been reduced by over one month, there might possibly have been an opportunity to actually conceive, implement and evaluate algorithms for information retrieval from implicitly crowdsourced data, but this can not be verified from the results available at this time.

On the brighter side, the architecture conceived proved to be same as efficient and scalable in comparison to any default web server, as the same constraints apply. Even further, it could be proven that the newly implemented components show the exact behaviour one would expect from any website running on an Apache httpd web server, neither enhancing nor worsening the performance and scalability of httpd.

Finally, the conceived concept can be applied not only for the MapBiquitous architecture. The conceived and implemented components allow a broad variety of applications, as the modularised structure actually allows any application logic to be deployed within the components. For example, the architecture conceived in this thesis allows replacement of the building servers with coffee machines, having the application logic implement controls for coffee brewing, of course using the Hyper Text Coffee Pot Control Protocol as described in RFC 2324[150].

9.2 FUTURE WORK

As this thesis and therefore inevitably also [Bom12] had to focus on the amendment of the existing MapBiquitous architecture in order to support crowdsourcing in a reliant way, only one actual use case – i.e. the correction of WLAN fingerprinting positions – could be considered. However, this use case could only be implemented on a per-user base; hence, the consolidation of submissions of different users remains unsolved. Therefore, the actual processing algorithm consolidating the separate results into one general result should be investigated in the future. Also, further applications implementing more use cases should be implemented in order to actually prove the new architecture's universal suitability for crowdsourcing.

With respect to the expansion of the MapBiquitous project to support community related aspects, the prepared database structures and interfaces could be focussed and the actual community provider could be considered. This is important as crowdsourcing relies on the crowd

[150] Of course, RFC 2324 is an IETF April Fool's joke. However, it can be utilised to show the possibilities of the concept conceived within this thesis.

being presented with a reward at least once in a while. Therefore, a clever reward system could be implemented in the context of the communities to be conceived. For example, special avatars could be made available only to prevalent crowdsourcers.

On an important note, the prepared signatures must be actually implemented. As for now, the MapBiquitous architecture envisages the use of signatures; however, only pseudo-signatures are implemented. These pseudo-signatures are generated and evaluated in method stubs, so the basic task in the future would be to actually implement an interface to the PKI and replace the stubs with actual application logic. Even further, currently the implementation does not pay any respect to security aspects such as genuineness of the INSANEs, etc. Therefore, excessive use of the PKI in order to use a key-based authentification could be considered.

For the moment, there are different INSANEs available, however they are not actually organised in a real DHT. All of the DHT functionality is simulated by method stubs and also partially missing. Hence, another thesis could focus on the implementation of the DHT-aspects of the INSANE distribution.

On a more technical note, support of IPv6[151] should be considered. Especially the current implementation uses the country-API of host-IP which only supports IPv4-based region lookup.

Finally, the current implementation only allows manual sifting of submissions using a direct database-access. The original concept of Gerd Bombach and the author of this thesis envisaged a trust-based automated sifting as well as manual sifting. Hence, a trust concept[152] as well as a sifting interface for the building managers should be investigated and implemented.

[151] Internet Protocol in version 6

[152] E.g. 'At least 20 submissions were made correcting this position and 17 of the submitters agree within a 5 metre radius; therefore, this 17-out-of-20 result can be automatically sifted'.

PART III APPENDIX

'If this is victory, then our hands are too small to hold it.'

from 'Lord of the Rings: The Return of the King' by John Ronald Reuel Tolkien, CBE

Contents of this Part

A MILESTONES

For the conceptual design and implementation of the proof-of-concept modifications to Map-Biquitous in line of the creation of this thesis as well as Gerd Bombach's assignment paper, the following milestones were scheduled:

- Milestone I – 13 August 2012
  - make database(s) consistent
    - merge POI data, fingerprint data and remaining building data into one building database per building
    - appoint a structure to new miscellaneous data such as semantics of room usage
    - create reasonable database structure for crowdsourcing data
  - make use of hashed IMEI for pseudo user-management
    - modify existing interfaces
    - determine and create new interfaces
    - create reasonable database-structure for user-management
    - for comparability: store all physical data on clients (anything involved in positioning, device type, WLAN and GSM module, CPU ('can the device participate in distributed computations should there be a later implementation of crowd computing?'))
    - store user consent on server (preparation for required signatures!), so that users can erase their submissions
  - GUI-Stub for client functionality
  - Tenshi: interim defence of the thesis
- Milestone II – 28 September 2012
  - manual correction of position
    - define and create new interface (get WGS-84 coordinate and create an new fingerprint there)
    - determine whether a new POI is created, or if an existing POI is being modified (must be done on server-side!)
    - define when a correction should be considered valid – idea: trust-value ('7 out of 10 say...')
    - determine when two correction-POIs are identical – precision of the detection or line of visibility of humans (ca. 3 metres)?
    - enrich existing fingerprints with GSM-information (Cell-ID, LAC strength, ...)
    - modify or create interfaces for new GSM-layer
  - required calculations on server-side
- Milestone III – 19 October 2012
  - verification and testing
  - evaluation and results

  - submission of written work
- Should there be excess time
  - further proof-of-concept implementation
  - map correction information – semantic information modifiable
  - divide MapBiquitous-client into a service for unaware crowdsourcing and a full-size GUI for the rest
    * outsource positioning and server-communication into service
    * rest (UI, calculations, etc.) into GUI
    * This is imperative! Sooner or later there is no way around this, so it should be at least discussed in 'future work'-section of the written work within Milestone III

Milestone I could not be met due to missing documentation and erroneous server interfaces in the geoserver, which was supposed to implement easy PostGIS-access. Further, incompatibility of Maven with the Android Development Kit prevented integration of the fingerprinting server into the per-building architecture in due time. Lastly, the interim defence was pushed to 3 September 2012.

Milestone II could not be met, as well. It had to be pushed to 8 October 2012 for the servers' side and 15 October 2012 for the clients' side, respectively. The aspects of validity and equality of submissions were removed from the milestone as these aspects could not be brought to a solution within the processing time. Rather, they seem to be good material for another student's assignment paper.

Milestone III was met with the exception of the submission of the written work. Printing of the written work was postponed to 22 October 2012 and the submission to 25 October 2012.

B GLOSSARY

API

Application Programming Interface; specification to be used as an interface for software components' communication; may include: routines (ready to use or their specification), data structures (ready to use or interfaces), object classes (ready to use or stubs) and variables (instantiated or instantiatable).

Dijkstra's algorithm

Algorithm to find the shortest route between two points (refer to Theorem D.0.1).

Framework

a collection of libraries and/or classes for a software (sub)system, intended for durable reusability.

HTTP

Hypertext Transfer Protocol; application protocol for distributed, collaborative, hypermedia information systems; foundation of browser-based communication in the World Wide Web; consists of multi-linear sets of objects, networking them by using logical (hyper)links between nodes.

MAC address

Media Access Control address; ideally unique identifier of network interfaces on physical network segment; normally assigned by hardware manufacturer, but modifiable; hence, not actually unique.

NFC

Near Field Communication; set of standards to establish close range (touch or touch-like proximity) wireless communication with each other.

NTP

Network Time Protocol; networking protocol used for synchronizing of clocks on distributed computer systems in data networks with variable latency.

OGC

Open Geospatial Consortium; international voluntary consensus standards organization encouraging development and implementation of open standards for geospatial content and services, geographic information data processing and data sharing.

POI

Point of Interest; specific point location considered useful or interesting; 'waypoint' may be used synonymously.

Recursion

process of self-similar repetition; 'In its most general numerical form the process of recursion consists in defining the value of a function by using other values of the same function' (Stanford Encyclopedia of Philosophy); if you still don't get it, see 'Recursion' in the glossary.

SHA

Secure Hash Algorithm; group of standardised cryptographic hash functions; calculate a (hopefully) unique hash value for any digital data.

SIM

Subscriber Identity/Identification Module; integrated circuit securely storing International Mobile Subscriber Identity (IMSI) with related key used to identify and authenticate mobile devices as subscribers in mobile networks.

Unification

let p and q be words or sentences over the same set, let $\mathbf{subst}(\mathbb{U}, p)$ be the result of applying substitution $\mathbb{U}$ on p and let $\mathbf{subst}(\mathbb{U}, q)$ be the result of applying substitution $\mathbb{U}$ on q ; does an $\mathbb{U}$ exist that holds $\mathbf{subst}(\mathbb{U}, p) = \mathbf{subst}(\mathbb{U}, q)$, then $\mathbf{unify}(p, q) := \mathbb{U}$ is solvable (using an unification algorithm) and p and q are unifiable, meaning they express the same structure over their set.

URL

Uniform Resource Locator; derivate of uniform resource identifier; specific character string that references to an Internet resource.

VPS

Virtual Private Server; a virtual machine, sharing the same physical server with other virtual machines; functionally is mostly equivalent to a separate physical computer.

W3C

World Wide Web Consortium; main international standards organization for the world wide web.

WGS-84

World Geodetic System (1984); standard for cartography, geodesy, and navigation; comprises a standard coordinate frame for the Earth, a reference ellipsoid, and a geoid defining the nominal sea level; used by the GPS; last revised in 2004.

XML

Extensible Markup Language; markup language defining a set of rules for document encoding; human-readable as well as machine-readable; defined in W3C's XML 1.0 Specification; gratis open standard.

C REFERENCES

[All12] ALLARD, L.: *Smart Power, Handy, Notruf – crowdmap 2 – | ffm-online*. 2012. – http://ffm-online.org/2012/05/30/smart-power-handy-notruf-crowdmap-2/
(cited on page 52)

[ASS+10] ALI, R. ; SOLIS, C. ; SALEHIE, M. ; OMORONYIA, I. ; NUSEIBEH, B. ; MAALEJ, W.: Social Sensing: When Users Become Monitors. In: *ACM D.2.2 [Softwareengineering]: Design Tools and Techniques* ESEC/FSE'11 (2010), 09. – ACM Code: 978-1-4503-0443-6/11/09
(cited on page 48)

[Bec12] BECKER, L.: Hackergruppe veröffentlicht eine Million iOS-Geräte-IDs. In: *heise online* (2012), 09. – http://www.heise.de/newsticker/meldung/Hackergruppe-veroeffentlicht-eine-Million-iOS-Geraete-IDs-1698373.html
(cited on pages 70 and 85)

[BKK+03] BALAKRISHNAN, H. ; KAASHOEK, M. F. ; KARGER, D. ; MORRIS, R. ; STOICA, I.: Looking Up Data in P2P Systems. In: *Communications of the ACM* 46 (2003), Nr. 2, S. 43–48
(cited on page 101)

[Bom12] BOMBACH, G.: *A title related to the MapBiquitous project*, Technische Universität Dresden, Assignment Paper (Belegarbeit), 11 2012. – published in parallel to this thesis; may currently be in print
(cited on pages 22, 37, 43, 70, 74, 82, 113, 123, 126, 141 and 191)

[Bun49] BUNDESREPUBLIK DEUTSCHLAND: *Bundesgesetzblatt*. Bd. 23.5.1949|1; 21.7.2010|944: *Grundgesetz für die Bundesrepublik Deutschland vom 23. Mai 1949 (BGBl. I S. 1) zuletzt geändert durch Gesetz vom 21. Juli 2010 (BGBl. I S. 944) mit Wirkung vom 27. Juli 2010*. 21.7.2010|944. Bundesrepublik Deutschland, 1949. – http://www.gesetze-im-internet.de/bundesrecht/gg/gesamt.pdf
(cited on page 71)

[Bun90] BUNDESREPUBLIK DEUTSCHLAND: *Bundesgesetzblatt*. Bd. 14.1.2003|66; 14.8.2009|2814: *Bundesdatenschutzgesetz in der Fassung der Bekanntmachung vom 14. Januar 2003 (BGBl. I S. 66), das zuletzt durch Artikel 1 des Gesetzes vom 14. August 2009 (BGBl. I S. 2814) geändert worden ist*. 14.8.2009|2814. Bundesrepublik Deutschland, 1990. – http://www.gesetze-im-internet.de/bundesrecht/bdsg_1990/gesamt.pdf
(cited on page 71)

[Bun04] BUNDESREPUBLIK DEUTSCHLAND: *Bundesgesetzblatt*. Bd. 22.6.2004|1190; 3.5.2012|958: *Telekommunikationsgesetz vom 22. Juni 2004 (BGBl. I S. 1190), das durch Artikel 1 des Gesetzes vom 3. Mai 2012 (BGBl. I S. 958) geändert worden ist*. 3.5.2012|958. Bundesrepublik Deutschland, 2004. – http://www.gesetze-im-internet.de/bundesrecht/tkg_2004/gesamt.pdf
(cited on page 71)

[Bun07] BUNDESREPUBLIK DEUTSCHLAND: *Bundesgesetzblatt*. Bd. 26.2.2007|179; 31.5.2010|692: *Telemediengesetz vom 26. Februar 2007 (BGBl. I S. 179), das zuletzt durch Artikel 1 des Gesetzes vom 31. Mai 2010 (BGBl. I S. 692) geändert worden ist*. 31.5.2010|692. Bundesrepublik Deutschland, 2007. – http://www.gesetze-im-internet.de/bundesrecht/tmg/gesamt.pdf
(cited on page 71)

[Bun12] BUNDESVERFASSUNGSGERICHT DER BUNDESREPUBLIK DEUTSCHLAND: *Beschluss des Ersten Senats vom 24. Januar 2012 – 1 BvR 1299/05*. court order, 1 2012. – https://www.bundesverfassungsgericht.de/entscheidungen/rs20120124_1bvr129905.html
(cited on page 72)

[DHP+12] DOBREV, M. ; HÖNACK, F. ; POHLE, M. ; SEIDENGLANZ, M. ; TITH, D.: *MapBiquitous II – Efficient Building Server Selection for a Mobile Indoor-Outdoor Location Service on Android*. 06 2012. – http://141.76.40.90/~wiki/index.php/MapBiquitousII
(cited on page 32)

[EG12] ESTELLÉS-AROLAS, E. ; GONZÁLEZ-LADRÓN-DE-GUEVARA, F.: Towards an integrated crowdsourcing definition. In: *Journal of Information Science* (2012), 02. – in press; DOI: 10.1177/016555150000000
(cited on page 41)

[GKN12] GRÜNEBERGER, F. J. ; KESSLER, C. ; NGUYEN, S. H.: *MapBiquitous – A mobile indoor/outdoor location service*. 06 2012. – http://141.76.40.90/~wiki/index.php/MapBiquitous
(cited on pages 32, 36 and 37)

[Gro] GROUP BMW: XFCD – Extended Floating Car Data Potenziale und Durchdringungsraten.
(cited on page 54)

[Gru12] GRUNAU, S.: *Konzeption und Evaluation erweiterter Mechanismen zur Positionsbestimmung in Gebäuden*, Technische Universität Dresden, Assignment Paper (Belegarbeit), 06 2012
(cited on pages 33, 51, 74, 85, 113, 126 and 213)

[How06] HOWE, J.: The Rise of Crowdsourcing. In: *WIRED magazine* 14.06 (2006), June. – DOI: 10.1145/1400181.1400200
(cited on page 41)

[Keß11] KESSLER, C.: *Konzeption und Umsetzung einer gebäudeübergreifenden Indoor/Outdoor-Navigation für das MapBiquitous System*, Technische Universität Dresden, Assignment Paper (Belegarbeit), 2011
(cited on page 32)

[Keß12] KESSLER, C.: *Entwicklung eines effizienten Konzeptes für die Verteilung der Verarbeitungslogik sowie das Management von Gebäudedaten in MapBiquitous*, Technische Universität Dresden, Thesis (Diplomarbeit), 03 2012
(cited on pages 32, 36, 37 and 38)

[Kir12] KIRSCH, C.: WhatsApp macht sich Authentifizierung leicht. In: *heise online* (2012), 09. – http://www.heise.de/security/meldung/WhatsApp-macht-sich-Authentifizierung-leicht-1703270.html
(cited on page 88)

[KSV11] KAUFMANN, N. ; SCHULZE, T. ; VEIT, D.: More than fun and money. Worker motivation in crowdsourcing – A study on mechanical turk. In: *Proceedings of the ...* (2011), Nr. 2009, S. 1–11. – http://schader.bwl.uni-mannheim.de/fileadmin/files/publikationen/Kaufmann_Schulze_Veit_2011_-_More_than_fun_and_money_Worker_motivation_in_Crowdsourcing_-_A_Study_on_Mechanical_Turk_AMCIS_2011.pdf
(cited on page 52)

[Lan07] LANDGERICHT BERLIN: *Urteil der Zivilkammer 23 vom 6. September 2007 – Geschäftsnummer 23 S 3/07 – 5 C 314/06 Amtsgericht Mitte*. court order, 9 2007. – http://www.daten-speicherung.de/data/Urteil_IP-Speicherung_2007-09-06.pdf
(cited on page 72)

[Mas11] MASLI, M.: Crowdsourcing Maps. In: *Computer* 44 (2011), 11, Nr. 11, 90-93. http://dx.doi.org/10.1109/MC.2011.338. – DOI 10.1109/MC.2011.338. – ISSN 0018–9162
(cited on pages 51 and 52)

[MCLP10] MADAN, A. ; CEBRIAN, M. ; LAZER, D. ; PENTLAND, A.: Social Sensing for Epidemiological Behavious Change. In: *ACM I.5.4 [Pattern Recognition]: Applications; ACM H.4.m [Information Systems]: Miscellaneous* UbiComp'10 (2010), 09. – ACM Code: 978-1-60558-843-8/10/09
(cited on page 48)

[MYS03] MILLER, M. S. ; YEE, K.-P. ; SHAPIRO, J.: Capability Myths Demolished. (2003), 02. – http://srl.cs.jhu.edu/pubs/SRL2003-02.pdf
(cited on page 95)

[NZZ11] NEIS, P. ; ZIELSTRA, D. ; ZIPF, A.: The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007-2011. In: *Future Internet* 4 (2011), 12, Nr. 1, 1-21. http://dx.doi.org/10.3390/fi4010001. – DOI 10.3390/fi4010001. – ISSN 1999–5903
(cited on page 51)

[OV05] OTSASON, V. ; VARSHAVSKY, A.: Accurate gsm indoor localization. In: *UbiComp 2005: ...* (2005), Nr. iv, S. 141–158. – http://www.springerlink.com/index/B6TBWEUG7QM91BWK.pdf
(cited on page 53)

[PMT10] PRIEDHORSKY, R. ; MASLI, M. ; TERVEEN, L.: Eliciting and focusing geographic volunteer work. In: *Proceedings of the 2010 ACM conference on Computer supported cooperative work – CSCW '10* (2010), 61. http://dx.doi.org/10.1145/1718918.1718931. – DOI 10.1145/1718918.1718931. ISBN 9781605587950
(cited on page 52)

[QB11] QUINN, A. J. ; BENDERSON, B. B.: Human Computation: A Survey and Taxonomy of a Growing Field. In: *ACM Conference on Human Factors in Computing Systems (CHI)*. Vancouver, BC, Canada, May 7-12 2011. – ACM Code: 978-1-4503-0267-8/11/05
(cited on page 41)

[Sch12] SCHMIDT, J.: US-Publishingdienstleister beichtet UDID-Leck. In: *heise online* (2012), 09. – http://www.heise.de/security/meldung/US-Publishingdienstleister-beichtet-UDID-Leck-1704070.html
(cited on page 70)

[Spr11] SPRINGER, T.: MapBiquitous – An Approach for Integrated Indoor/Outdoor Location-based Services. (2011)
(cited on pages 32 and 33)

[STW02] SCHÄFER, R. P. ; THIESSENHUSEN, K. U. ; WAGNER, P.: A traffic information system by means of real-time floating-car data. In: *ITS world congress* m (2002), S. 1–8. – `http://elib.dlr.de/6499/01/chicago_final.pdf`
(cited on page 54)

[Tel07] TELHAN, O.: *Social Sensing and Its Display*, Massachusetts Institute of Technology, Master Thesis, 08 2007. – Bilkent Univeristy, Ankara
(cited on page 48)

[Wer12] WERNER, K.: *Framework zur Implementierung von Indoor/Outdoor Location-based Services mit MapBiquitous*, Technische Universität Dresden, Thesis (Diplomarbeit), 03 2012
(cited on page 32)

C.1 AUXILIARY MEANS

While creating this thesis, the following auxiliary means were used in order to type-set and research the written submission, as well as to implement the proof-of-concept implementation:

- Adobe Acrobat Pro X (10.1.4)
- Adobe Dreamweaver CS5 (11.0.4993)
- Brockhaus Enzyklopädie, 21. Auflage
- Comprehensive TeXArchive Network, The – `http://ctan.org`
- Corel Draw PhotoPaint X5 (15.2.0.695)
- dict.leo.org by LEO GmbH (dictionary and online translator)
- Eclipse Indigo and Eclipse Juno as well as their reference guides
- Encyclopædia Britannica, Digital Version
- friends and acquaintances (proofreading)
- Google-Scholar (search-engine)
- Microsoft Office Excel 2010
- Microsoft Office Visio 2010
- Microsoft Office Word 2010
- MiKTeX 2.9 (LaTeX typesetting-tool for Windows)
- Notepad++ 6.2 (editor)
  with the following run-dialog on *Alt+F8* it is very useful:
  ```
  cmd /c cd /d "$(CURRENT_DIRECTORY)"
     && texify "$(FILE_NAME)"
     && bibtex "$(NAME_PART)"
     && texify --clean --pdf --run-viewer "$(FILE_NAME)"
  ```
- OriginLab Origin 6
- Paint.NET 3.5.10
- QR-Code generator by Dominik Dzienia (©2010) – 'QR-Code' or 'Quick Response Code' is a technology and registered trademark of Denso Wave and is subject to patent protection; Denso Wave has release QR-Code as open source to free usage
- Sächsische Landesbibliothek – Staats- und Universitätsbibliothek Dresden
- Wikipedia, German and English (finding some references)
- Wiktionary (dictionary)
- Wireshark 1.8.3
- XAMPP 1.7.7 with Apache/2.2.21 (Win32), mod_ssl/2.2.21, OpenSSL/1.0.0e, PHP/5.3.8, mod_perl/2.0.4, Perl/v5.10.1 and mysqlnd 5.0.8-dev-20102224 (Revision 310735)

ACKNOWLEDGEMENTS

Creating this thesis did not only mean stress for me, but – to a certain degree – also for my surroundings, my wife, my friends and co-workers in particular. Therefore, some kind words of acknowledgement should not be held back.

Firstly, I want to thank my wife Karina. Without her persevering endurance, her support, her advice and her presence each time I needed consolation, I would have never been able to finish this Thesis. Especially not over a month before the submission deadline. (~~)(@)<3

Definitively an acknowledgement has to be directed to my parents for varying reasons, but I do not know what to write to express my gratitude. Mama & Papa, you know what I want to tell you even when it can't be expressed by written words.

Mr. Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill shall be thanked for enabling the swift creation of this thesis within four and a half month, rather that six month. Who would not jump at a job offer by their professor presented just at the beginning of the processing time of their thesis?

A very special acknowledgement is directed towards my supervising tutor Mr. Dr.-Ing. Thomas Springer, who fitted this Thesis' topic towards my personal field of interest and supported the earlier submission.

Another special acknowledgement is dedicated to Mr. Gerd Bombach. Due to the cohesion in the tasks of his assignment paper (Belegarbeit) to my thesis, we had to closely work together, not only sharing ideas and concepts, but also stress due to my shortened processing time. Gerd, I am very sorry for all the stress and pressure of time I have given you.

Stephan Hermsdorf deserves acknowledgement for his work. Everybody takes an administrator for granted, but in reality, the entire IT would not work without them. Therefore, I wish to express my gratitude for Stephan's good work and his swift support, especially when I requested more server resources, another virtual machine for testing, etc.

I wish to emphasise the acknowledgement for my proofreaders; without their feedback this Thesis would most definitely look totally different... Or at least it would be very hard to read. – These busy readers are: Mr. Dipl.-Phys. Ronald Stübner, Mr. Gregor Tomaszewski, B.Sc., my wife and for some part my supervising tutor.

Finally, I want to thank the Chair of Computer Networks at the TU Dresden for all the possibilities they gave me during the time I spent creating this Thesis.

PART IV SUPPLEMENTARY

Contents of this Part

Figures within this Part

Tables within this Part

D PROOFS AND DEFINITIONS

Theorem D.0.1 – Dijkstra's algorithm

Let $\mathbb{G} = \{V, E\}$ be a directed graph with
- vertices $\{v_1, v_2, \ldots, v_n\} \in V$ and
- edges with lengths $\left\{\left(v_i, v_j, l\right) \mid i \neq j \mid l \geq 0 \mid v_i \text{ connected to } v_j\right\} \in E$.

Further, let v_1 be the starting vertex and let v_n be the (final) destination vertex. Then the algorithm shall work as follows:

1. Mark all vertices *white* and pool them in a set $\mathbb{W} = \{v_1, 0\} \cup \bigcup_{\lambda=2}^{n} \{(v_\lambda, \infty)\}$ (the set of unvisited vertices).
2. Create an empty set $\mathbb{B}$ for vertices marked *black* (the set of visited vertices).
3. From the set of unvisited vertices, extract the vortex with the smallest tentative distance $\neq \infty$ and reduce $\mathbb{W}$ by that vortex. Mark the extracted vortex as being *grey*. Iff $\mathbb{W}$ only consists of vertices with tentative distances $= \infty$, terminate the algorithm with an 'unreachable final destination' error.
4. For the vortex currently marked *grey*, consider all of its *white* neighbours (i.e. $\in \mathbb{W}$). Calculate the tentative distances from v_1 to the *white* neighbours using the lengths from the set E. Iff a calculated distance of a vortex is less than the previously recorded tentative distance of the same vortex in the set $\mathbb{W}$, then overwrite that distance.
5. When all of the *white* neighbours of the vortex currently marked *grey* have been considered, mark the vortex currently marked *grey* (let this be v_{grey}) as being *black* and reduce $\mathbb{W} = \mathbb{W} \setminus \left\{\left(v_{\text{grey}}, t\right)\right\}$, $t \in \mathbb{N}$. Add the vortex newly being marked *black* to the set of *black* vertices: $\mathbb{B} = \mathbb{B} \cup \left\{v_{\text{grey}}\right\}$.
6. If the destination vortex v_n has been marked *black* terminate the algorithm and output the tentative distance of v_n as final result (i.e. the shortest route from v_1 to v_n), otherwise repeat from step 3.

Proof D.0.2 – Dijkstra's algorithm

- For a graph with no vertices or only one vortex the algorithm is not applicable.
- For a graph with two vertices:
  - Should there be no connecting edge, the algorithm outputs an 'unreachable final destination' error and terminates correctly.
  - Should there be at least one connecting edge, the algorithm outputs the shortest edge's length as tentative length and terminates correctly.
- For each vortex added to the set of vertices:
  - Should the new vortex be the final destination:

    - Should there be no edge connecting to the new vertex, the algorithm outputs an ‘unreachable final destination’ error and terminates correctly.
    - Should there be at least one connecting edge, the algorithm calculates a new tentative length for the new vertex, adding each connecting edge’s length to each already calculated tentative length of the previously existing vertices. After finishing these calculations, the algorithm outputs the new vortex’s tentative length and terminates correctly.
  - Should the new vortex not be the final destination:
    - Should there be no edge connecting to the new vortex, the algorithm maintains the tentative length ∞ for the new vortex over the entire run of the algorithm.
      - Should there be one or more edges connecting to the final destination only from the new vortex, the algorithm will output an ‘unreachable final destination’ error and terminates correctly.
      - Should there be edges connecting to the final destination from other vertices but the new vortex, the algorithm might terminate either by outputting a length, or by outputting an ‘unreachable final destination’ error. The output depends on the structure of the existing edges, but the algorithm will terminate correctly either way.
    - Should there be at least one connecting edge to the new vortex, the algorithm calculates a new tentative length for the new vortex, adding each connecting edge’s length to each already calculated tentative length of the previously existing vertices. After finishing these calculations, edges connecting from the new vortex will be considered:
      - Should there be one or more edges connecting to the final destination, the algorithm will output the shortest tentative distance and terminates correctly.
      - Should there be edges connecting to the final destination from other vertices but the new vortex, the algorithm might terminate either by outputting a length, or by outputting an ‘unreachable final destination’ error. The output depends on the structure of the existing edges, but the algorithm will terminate correctly either way.

As the algorithm is either not applicable or terminates correctly with a distance or an error for any number of vertices, Theorem D.0.1 is valid.

▪

E CODE SNIPPETS

E.1 SCHEDULED TASK ON CARLOS

This simple command line batch script restarts Apache Tomcat and PostgreSQL via Windows' services handler (`net services`). In the current CARLOS configuration, it is set to run every day at 04:00h CET.

Programme E.1.1: Simple command line batch file restarting Tomcat and PostgreSQL

```
@echo off
echo.
echo ####################################################################
echo # This is a scheduled restart of PostgreSQL and Apache Tomcat. #
echo ####################################################################
echo.
echo.
echo Attempt: Stopping PostgreSQL...
net stop "postgresql-9.0 - PostgreSQL Server 9.0"
echo Done. - Waiting 3 seconds before restart.
timeout 3
echo.
echo Attempt: Restarting PostgreSQL...
net start "postgresql-9.0 - PostgreSQL Server 9.0"
echo Done.
echo.
echo Waiting 20 seconds before stopping Tomcat.
timeout 20
echo.
echo Attempt: Stopping Apache Tomcat...
net stop "Apache Tomcat 6"
echo Done. - Waiting 3 seconds before restart.
timeout 3
echo.
echo Attempt: Restarting Apache Tomcat...
net start "Apache Tomcat 6"
echo Done.
echo.
echo Scheduled restarts finnished.
```

E.2 HTTP-HANDLER MODULE IN THE INSANE AND BSCSM

The `HTTP_Requests` module is a handler module, encapsulating all POST and GET method based HTTP communication on socket level. As all communication within MapBiquitous is either POST or GET method based HTTP communication, the implemented handler module only supports these two types of HTTP methods, but it is extensible to support more methods. However, other method types (e.g. PUT) do not require support via this handler module as they are used by none of the MapBiquitous components, and future components can be designed to be based on POST or GET. Additionally, a `ping` function is provided as system calls to the target systems' `ping` function might be blocked in PHP safemode.

Programme E.2.1: The HTTP handler module in the INSANE and BSCSM

```
function ping($host, $port=80, $time=2) {
    $socket = fsockopen($host, $port, $errorNumber, $ErrorString, $time);
    if (!$socket) return false;
    fclose($socket);
    return true;
}

function post_request($data, $url="127.0.0.1:8080", $secure=true) {
    global $_CONFIG;
    #
    # The following HTTP-context MUST contain the ignore_errors-switch
    # set to "true", as fopen does not return the response-body if the
    # HTTP-status is different from "200 OK". Some methods do actally
    # return "409 Conflict" for deprecated or "501 Not Implemented"
    # for stub methods. However, such HTTP-packets may still contain
    # results in their body that are required to continue computation.
    #
    $params =   array(  'http' =>   array(
                            'method'        => 'POST',
                            'content'       => http_build_query($data),
                            'ignore_errors' => true));
    if ($secure && $_CONFIG["allowSelfSigned"])
        $params['ssl'] = array('allow_self_signed' => true);
    $context = stream_context_create($params);
    $connect = @fopen("http".($secure?"s":"").":".$url,
                      'rb', false, $context);
    $status = $http_response_header;
    $status = $status[0];
    $status = substr($status, 9, 3);
    switch ($status) {
        case "200": # OK
                    $response = @stream_get_contents($connect);
                    $meta = @stream_get_meta_data($connect);
                    if ($response === false)
                        throw new Exception(
                            "Error while retrieving response:
                              ".$php_errormsg,
                            200);
                    return array($meta, $response);
                    break;
        case "202": # Accepted
                    throw new Exception(
                        "The contacted host accepted the request.",
                        202);
                    break;
        #
        # Et cetera for errors 302, 303, 400, 403, 404, 409, 500, 501,
```

```
47          # 502 and 503.
48          #
49          default:    throw new Exception(
50                          "The contacted host returned an unexpected code.",
51                          $status);
52                      break;
53      }
54  }
55
56  function get_request($data, $url="127.0.0.1:8080", $secure=true) {
57      global $_CONFIG;
58      #
59      # The following HTTP-context MUST contain the ignore_errors-switch
60      # set to "true", as fopen does not return the response-body if the
61      # HTTP-status is different from "200 OK". Some methods do actally
62      # return "409 Conflict" for deprecated or "501 Not Implemented"
63      # for stub methods. However, such HTTP-packets may still contain
64      # results in their body that are required to continue computation.
65      #
66      $params =   array('socket' => array('bindto' => '0:'.$port),
67                      'http' => array('method' => 'GET',
68                                      'ignore_erros' => true));
69      if ($secure && $_CONFIG["allowSelfSigned"])
70          $params['ssl'] = array('allow_self_signed' => true,);
71      $context = stream_context_create($params);
72      $getparams = "";
73      foreach($data as $key=>$val) $getparams .= $key."=".$val."&";
74      $connect = @fopen("http".($secure?"s":"").":".$url."?".$getparams,
75                          'rb', false, $context);
76      $status = $http_response_header;
77      $status = $status[0];
78      $status = substr($status, 9, 3);
79      switch ($status) {
80          case "200": # OK
81                      $response = @stream_get_contents($connect);
82                      $meta = @stream_get_meta_data($connect);
83                      if ($response === false)
84                          throw new Exception(
85                              "Error while retrieving response:
                                    ".$php_errormsg,
86                              200);
87                      return array($meta, $response);
88                      break;
89          case "202": # Accepted
90                      throw new Exception(
91                          "The contacted host accepted the request.",
92                          202);
93                      break;
94          #
95          # Et cetera for errors 302, 303, 400, 403, 404, 409, 500, 501,
96          # 502 and 503.
97          #
98          default:    throw new Exception(
99                          "The contacted host returned an unexpected code.",
100                         $status);
101                     break;
102     }
103 }
```

E.3 GETCOUNTRY() AND ASSOCIATED FUNCTIONS

This basic function simply sends the client's IP-address to hostip.info's country API and returns the result of the API request. Though, it requires a few helper functions.

Programme E.3.1: Simple PHP function to retrieve an IP's geographical region (country)

```
1  function countryIP() {
2      #
3      # The returned result should be a two-letter all-caps country-code.
4      #
5      return safeLoad("api.hostip.info", "/country.php?ip=".getIP());
6  }
```

The `getIP()` function basically tries to determine a client's actual IP-address, even when using a proxy. For this, a row of non-default `$_SERVER[]` variables are tested in lines 9, 10, 15, 18 and 21. As these variable are not included in the PHP-defaults, it depends on the server's configuration whether these environment variables are set or not. Should they or parts of them exist, which also depends on whether the client's proxy transmits proxy headers, the corresponding value is returned, iff the value is a valid IP-address (determined by the `validIP()` function). Otherwise, the default `$_SERVER["REMOTE_ADDR"]` is returned.

Programme E.3.2: PHP function to retrieve a client's IPv4-address

```
1  function getIP() {
2      #
3      # Note: Depending on PHP-versions, some servers are not able to handle
4      #       forwarding-headers; hence, an isset()-test needs to be added to
5      #       the non-standard header-fields.
6      #
7      if (isset($_SERVER["HTTP_CLIENT_IP"])
8          && (validIP($_SERVER["HTTP_CLIENT_IP"])))
9          return $_SERVER["HTTP_CLIENT_IP"];
10     if (isset($_SERVER["HTTP_X_FORWARDED_FOR"]))
11         foreach (explode(",",$_SERVER["HTTP_X_FORWARDED_FOR"]) as $ip)
12             if (validIP(trim($ip))) return $ip;
13     if (isset($_SERVER["HTTP_X_FORWARDED"])
14         && validIP($_SERVER["HTTP_X_FORWARDED"]))
15         return $_SERVER["HTTP_X_FORWARDED"];
16     elseif (isset($_SERVER["HTTP_FORWARDED_FOR"])
17         && validIP($_SERVER["HTTP_FORWARDED_FOR"]))
18         return $_SERVER["HTTP_FORWARDED_FOR"];
19     elseif (isset($_SERVER["HTTP_FORWARDED"])
20         && validIP($_SERVER["HTTP_FORWARDED"]))
21         return $_SERVER["HTTP_FORWARDED"];
22     else return $_SERVER["REMOTE_ADDR"];
23 }
```

Basically, the `validIP($ip)` function checks whether a string (`$ip`) given to the function is actually an IPv4-address, and whether it is not within a reserved, non-public IPv4-address range.

Programme E.3.3: PHP function to check whether a string is a valid IPv4-address

```
1  function validIP($ip) {
2      if (!empty($ip) && ip2long($ip)!=-1) {
3          $reserved_ips = array (
4              array('0.0.0.0','0.255.255.255'),
```

```
            array('10.0.0.0','10.255.255.255'),
            array('127.0.0.0','127.255.255.255'),
            array('169.254.0.0','169.254.255.255'),
            array('172.16.0.0','172.31.255.255'),
            array('192.0.2.0','192.0.2.255'),
            array('192.168.0.0','192.168.255.255'),
            array('255.255.255.0','255.255.255.255'));
        foreach ($reserved_ips as $r) {
            $min = ip2long($r[0]);
            $max = ip2long($r[1]);
            if ((ip2long($ip) >= $min) && (ip2long($ip) <= $max))
                return false;
        }
        return true;
    }
    else return false;
}
```

Finally, the actual connection to the hostip.info country API is realised using the `safeLoad($domain, $path, $timeout)` function. Its basic task is to ensure that PHP does not get stuck during a connection attempt; hence, a socket connection is used to verify target-reachability before actually conducting a HTTP-based `file_get_contents($path)` call to the desired resource. Should the socket connection fail, a correct Boolean `false` is returned.

Programme E.3.4: PHP function to retrieve a WWW resource safely

```
function safeLoad($domain, $path, $timeout = 10) {
    #
    # This function ensures that the remote host is actually reachable.
    # Should there be a problem, the function returns Boolean false.
    #
    $fp = fsockopen($domain, 80, $errno, $errstr, $timeout);
    if($fp) {
        $out = "GET ".$path." HTTP/1.1\r\n";
        $out .= "Host: ".$domain."\r\n";
        $out .= "Connection: Close\r\n\r\n";
        fwrite($fp, $out);
        $resp = "";
        while (!feof($fp))
            $resp .= fgets($fp, 128);
        fclose($fp);
        $status_regex = "/HTTP\/1\.\d\s(\d+)/";
        if (preg_match($status_regex, $resp, $matches)
            && ($matches[1] == 200)
        ) {
            $content = file_get_contents("http://".$domain.$path);
            if ($content) return $content;
            return false;
        }
    }
    return false;
}
```

E.4 JSON_HANDLER.PHP

This module handler allows easy replacement of the JSON-handling functions of the INSANEs and BSCSMs, thus making it possible to circumvent shortcomings of older PHP versions while

maintaining usability of the advantages of newer PHP versions. To be more specific, the underlying `JSON_Library.php` provides the `json_encode($structure)` and `json_decode($json)` functions under PHP 4.x through 5.1.x. In PHP 5.2.x and newer these functions are integrated into PHP.

Programme E.4.1: The JSON-handling module of the INSANEs and BSCSMs

```
1  <?php
2  if (!function_exists('json_encode')) {
3      include_once(getcwd()."/JSON_Library.php");
4      $GLOBALS['JSON_OBJECT'] = new Services_JSON();
5      function json_encode($value) {
6          return $GLOBALS['JSON_OBJECT']->encode($value);
7      }
8      function json_decode($value) {
9          return $GLOBALS['JSON_OBJECT']->decode($value);
10     }
11 }
12
13 function isJSON($string) {
14     try{
15         $structure = json_decode($string);
16     }
17     catch(Exception $e){
18         return false;
19     }
20     return (is_object($structure)) ? true : false;
21 }
22
23 function fromJSON($json) {
24     if (isJSON($json)) return json_decode($json, true);
25     return false;
26 }
27
28 function toJSON($structure) {
29     return json_encode($structure);
30 }
31 ?>
```

E.5 EXEMPLARY DEFINITION BLOCK

Incidentally, modularising the INSANE as well as BSCSM source code while maintaining 'human readability' of the methods offered required some crafty definitions of scanable syntax. In the end, each method can be automatically loaded into the running INSANE or BSCSM by strictly following the newly created *definition block syntax*, as it is completely parsable[153] by PHP. The structure is rather simple, relying on keywords such as '`DEFINITION BLOCK`' or '`@return`', delimiters such as '`||`' or '`|||`', as well as PHP's own serialisation syntax, allowing automated usage of PHP's `serialize` and `deserialize` functions. An exemplary method definition block using the defined syntax is shown in Programme E.5.1.

[153]There are controversial discussions dealing with the existence of the noun 'parsability' and the corresponding adjective 'parsable' (or 'parseable'). However, the author of this thesis assumed the adjective to exist. Should it not exist, the meaning defined by Noam Chomsky shall be valid: 'The ability to assign a structural analysis to sentence'.

Programme E.5.1: An exemplary method definition block

```php
#
# Define the output type.
#
header("Content-Type: text/plain");
#
# DEFINITION BLOCK
#
# @stub|||The following statement is either "true" or "false"
$isStub = true;
if ($isStub) header("HTTP/1.1 501 Not Implemented");
#
# @dep|||The following statement either "true" or "false"
$isDep = false;
if ($isDep) {
    header("HTTP/1.1 409 Conflict");
    print("This method is deprecated!");
    exit;
}
#
# @desc|||A descriptive text (HTML allowed)
# @input|||a:2:{i:0;s:38:"(Type)||4||var1||Variable 1 (length
   4)";i:1;s:38:"(Type)||9||var2||Variable 2 (lenth 9)";}
# @return|||a:1:{i:0;s:30:"HTTP/1.1 200 OK||Normal output";}
# @throws|||a:2:{i:0;s:79:"HTTP/1.1 400 Bad Request||The submitted POST
   data do not match the expectations";i:1;s:50:"HTTP/1.1 502 Bad
   Gateway||A relaying error occurred";}
# @author|||Tenshi Hara [hara@inf.tu-dresden.de]
# @date|||2012-10-04
#
# END OF DEFINITION BLOCK
#
################################################################################
################################################################################
################################################################################
#
# Define expected variables
#
$expected_variables = array(
    "method", # this one is a global variable and required!
    "var1",
    "var2");
sort($expected_variables);
#
# Automatically check whether the submitted variables match
# the expected variables.
#
#   ########################
#   # MODULE LOADING BLOCK #
#   ##################################################
#   Load any module required for processing here     #
require(getcwd()."/methods/Submission_Checker.php"); #
#   ##################################################
#   # Make sure these modules exist as require() #
#   # halts the PHP execution if the module file #
#   # could not be included into the runtime.    #
#   ##############################################
#
# THIS TEST NEEDS TO BE DEFINED MANUALLY!
#
#   ############################################################################
```

```
58  #   # In this example, there are no formal requirements to the variables, #
59  #   # but should there be some, they should be checked here.               #
60  #   ##########################################################################
61  #
62  if ((strlen($method_values["var1"]) > 4)
63      || (strlen($method_values["var2"]) > 9)
64  ) {
65      header("HTTP/1.1 400 Bad Request");
66      if ($_CONFIG["debug"]) {
67          print("Error in line ".__LINE__.": ".
68              ((strlen($method_values["var1"])>4)?"(var1 too long)":"").
69              ((strlen($method_values["var2"])>9)?"(var2 too long)":"")
70          );
71      }
72      flush();
73      exit;
74  }
75  #
76  # Starting from this line, the actual module source code follows.
77  #
```

The automatically processed definition block part used to generate human readable method descriptions starts in line 6 and ends in line 27. It contains information about whether the method created by the module is a stub, whether it is deprecated, its input and return values, as well as errors to be expected to be thrown. It should be noted that the list of thrown errors normally should only contain errors that are expectable and generated during the course of the computation. Possible but unexpected errors (such as 'HTTP/1.1 500 Internal Server Error') should not be listed within the '`@throws`' serialisation. For the automated check of variable completeness[154], the set of expected variables is defined in line 35. It is automatically processed in the '`Submission_Checker`' module that is bound in line 48, resulting in the creation of the array `$method_values` which is e.g. used in the manual check of compliance with formal requirements towards the input variables in line 62. The automated processing – that basically consists of de-serialisations and loops – can follow Programme E.5.2.

Programme E.5.2: An exemplary parser for method definition blocks

```
1   #
2   # Assume the contents of the method file are in $filecontents...
3   #
4   $methodIsStub = strpos($filecontents, "isStub = true;");
5   $methodIsDep = strpos($filecontents, "isDep = true;");
6   $filecontents = substr($filecontents,
7                       strpos($filecontents, "# @desc"),
8                       strpos($filecontents, "# END OF DEFINITION BLOCK")-39);
9   $filecontents = explode("
10  # @", $filecontents);
11  function arraycut($value) return substr($value, strpos($value, "|||")+3);
12  $filecontents = array_map('arraycut', $filecontents);
13  #
14  # $filecontents now is an array of serialised arrays
15  #
16  # Load input variables:
17  $filecontents[1] = unserialize($filecontents[1]);
18  foreach ($filecontents[1] as $inputvar) {
19      $inputvar = explode("||", $inputvar);
20      # Do something with the input variables...
21      # Note: $inputvar is an array having
22      #         * $inputvar[0]: the type of the variable
23      #         * $inputvar[1]: the maximum length of the variable
```

[154] I.e. if all variables expected as input are present in the POST header of the HTTP request calling the method.

```
24      #           * $inputvar[2]: the name of the variable
25      #           * $inputvar[3]: the variable's description
26  }
27  #
28  # Load return values:
29  $filecontents[2] = unserialize($filecontents[2]);
30  foreach ($filecontents[2] as $returns) {
31      $returns = explode("||", $returns);
32      # Do something with the return values...
33      # Note: $returns is an array having
34      #           * $returns[0]: the return value
35      #           * $returns[1]: the return value's description
36  }
37  #
38  # Load the throwable errors:
39  $filecontents[3] = unserialize($filecontents[3]);
40  foreach ($filecontents[3] as $throws) {
41      $throws = explode("||", $throws);
42      # Do something with the throwable errors...
43      # Note: $throws is an array having
44      #           * $throws[0]: the thrown error
45      #           * $throws[1]: the thrown error's description
46  }
47  #
48  # The author of the module is in $filecontents[4]
49  #
50  # Load the date of the module:
51  $filecontents[5] = substr($filecontents[5], 0, (strpos($filecontents[5], "
52  ") > 0) ? strpos($filecontents[5], "
53  ") : strlen($filecontents[5]));
```

F TABLES

F.1 FINGERPRINTING OVERHEAD RESULTS

Table F.1.1: Results for Fingerprinting Submissions in %

Samples↔INSANE	Overhead Client↔INSANE	Total Overhead
1	22.59 (173.54)	173.80 (475.71)
2	12.41 (123.11)	145.82 (367.21)
3	6.52 (93.91)	129.62 (304.40)
4	2.68 (74.87)	119.06 (263.44)
5	−0.02 (61.48)	111.62 (234.61)
6	−2.02 (51.53)	106.11 (213.23)
7	−3.57 (43.87)	101.85 (196.73)
8	−4.80 (37.77)	98.47 (183.62)
9	−5.80 (32.81)	95.72 (172.94)
10	−6.63 (28.69)	93.44 (164.09)
20	−10.74 (8.35)	82.15 (120.32)
30	−12.25 (0.82)	77.97 (104.12)
50	−13.53	74.46

continued on next page ⇝

⇝ continued from previous page

Samples↔INSANE	Overhead Client↔INSANE	Total Overhead
	(–5.51)	(90.50)
100	–14.53 (–10.45)	71.72 (79.88)

Results are based on actual communication using the INSANE at `insane.the-tester.de`.

F.2 POSITIONING OVERHEAD RESULTS

Table F.2.1: Results for Positioning in %

Samples↔INSANE	Overhead Client↔INSANE	Total Overhead
1	56.75 (257.44)	221.11 (622.49)
2	48.24 (218.82)	202.94 (544.12)
3	41.94 (190.28)	189.51 (486.19)
4	37.10 (168.33)	179.19 (441.63)
5	33.27 (150.91)	170.99 (406.29)
6	30.15 (136.76)	164.34 (377.57)
7	27.56 (125.04)	158.82 (353.78)
8	25.39 (115.17)	154.18 (333.75)
9	23.53 (106.74)	150.22 (316.64)
10	21.93 (99.47)	146.79 (301.87)
20	13.04 (59.14)	127.82 (220.03)
30	9.28 (42.08)	119.80 (185.41)
50	5.88 (26.69)	112.55 (154.16)
100	3.07 (13.94)	106.56 (128.29)

Results are based on actual communication using the INSANE at `insane.the-tester.de`.

F.3 HARDWARE USED FOR PERFORMANCE AND SCALABILITY TESTS

Table F.3.1: The server hardware configuration of the evaluation

	BSCSMs & INSANE on 'CARLOS'	INSANE in LAN	the-tester.de	hara.tc
Type	shared	dedicated	Cloud-VPS	Cloud-VPS
CPU-Type	Xeon E5620	Core 2 Duo L7500	n/a (VPS)	n/a (VPS)
CPU-Speed	up to 2.4 GHz	up to 1.6 GHz	up to 3.0 GHz	up to 3.0 GHz
RAM	2 GB	2 GB	up to 4 GB	up to 4 GB
Operating System	MS Win 5.2.3790, x32	MS Win 5.2.2600.5512, x32	Linux 2.6.32-41-server, x64	Linux 2.6.32-41-server, x64
WWW-server	Apache httpd 2.2.14	Apache httpd 2.2.21	Apache httpd 2.0.?	Apache httpd 2.0.?
Script-Language	PHP 5.2.6	PHP 5.3.8	PHP/5.2.12-nmm4	PHP/5.2.12-nmm4
Database	MySQL 5.1.?	MySQL 5.0.8	MySQL 5.1.63	MySQL 5.1.63
Downlink	1 $\frac{GBit}{s}$	500 $\frac{MBit}{s}$	60 $\frac{MBit}{s}$	60 $\frac{MBit}{s}$
Uplink	< 600 $\frac{MBit}{s}$	500 $\frac{MBit}{s}$	55 $\frac{MBit}{s}$	55 $\frac{MBit}{s}$
Avg. Idling Load	≥ 0.6[155]	< 0.1	< 0.1	< 0.1

Information are provided as available.

Table F.3.2: The client hardware configuration of the evaluation

	Host 1	Host 2	Host 3
Type	1024 virtual devices	512 virtual devices	8976 virtual devices
CPU-Type	Core i7-2640M	Core 2 Duo L7500	n/a (VPS)
CPU-Speed	up to 2.8 GHz	up to 1.6 GHz	up to 3.0 GHz
RAM	8 GB	2 GB	up to 4 GB
Operating System	MS Win 6.1.7601, x64	MS Win 5.2.2600.5512, x32	Linux 2.6.32-41-server, x64
Simulator-Language	C#	C#	Perl
Downlink	5880 $\frac{kBit}{s}$	5880 $\frac{kBit}{s}$	60 $\frac{MBit}{s}$
Uplink	192 $\frac{kBit}{s}$	192 $\frac{kBit}{s}$	55 $\frac{MBit}{s}$
Avg. Script-Load	≥ 0.92	≥ 0.97	≥ 0.85

Information are provided as available.

[155] It remains unconceivable why 'CARLOS' has such a high load while idling.

F.4 PERFORMANCE AND SCALABILITY RESULTS

Table F.4.1: Performance & Scalability Results

Performance & Scalability Results												
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	'200 OK' sent by INSANE	'200 OK' received by Client by Client	'502 Bad Gateway' sent by INSANE	'502 Bad Gateway' received by Client	Reply timed out on Client
1	1	1	1	1	1	1	1	1	1	0	0	0
5	5	5	5	5	5	5	5	5	5	0	0	0
10	10	10	10	10	10	10	10	10	10	0	0	0
50	50	50	50	50	50	50	50	50	50	0	0	0
100	99	99	98	98	98	99	99	98	98	1	1	1
500	497	497	494	494	491	497	494	491	488	6	6	6
1000	994	994	988	988	982	994	988	982	977	12	11	12
5000	4962	4962	4924	4924	4887	4962	4924	4887	4862	75	62	76
10000	5109	5109	5068	5068	5027	5109	5068	5027	5001	82	67	4932

All numbers are averaged of ten series of measurements.

F.5 UNACCOUNTED-FOR PERFORMANCE AND SCALABILITY RESULTS

F.5.1 WFS-Series

Table F.5.1: WFS results for the LAN setting

LAN Setting									
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	'200 OK'	'502 Bad Gateway'
1	1	1	1	1	1	1	1	1	0
5	5	5	5	5	5	5	5	0	
10	10	10	10	10	10	10	10	10	0
50	50	50	26	26	26	50	50	13	37
100	100	100	49	49	49	100	100	24	76
500	500	500	187	187	187	500	500	70	430
1000	1000	1000	343	343	343	1000	1000	117	883
5000	5000	5000	1312	1312	1312	5000	5000	344	4656
10000	10000	10000	2401	2401	2401	10000	10000	576	9424

All numbers are averaged of ten series of measurements.

Table F.5.2: WFS results for the 'CARLOS' setting

the-tester.de setting									
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	'200 OK'	'502 Bad Gateway'
1	1	1	1	1	1	1	1	1	0
5	5	5	5	5	5	5	5	5	0
10	10	10	10	10	10	10	10	10	0
50	26	26	25	25	24	26	26	24	2
100	49	49	48	48	47	49	49	47	2
500	187	187	181	181	176	187	166	156	10
1000	343	343	332	332	322	343	299	281	18
5000	1312	1312	1263	1263	1216	1312	1090	1010	80
10000	2401	2401	2306	2306	2215	2401	1962	1810	152

All numbers are averaged of ten series of measurements.

Table F.5.3: WFS results for the the-tester.de setting

'CARLOS' setting									
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	'200 OK'	'502 Bad Gateway'
1	1	1	1	1	1	1	1	1	0
5	5	5	5	5	5	5	5	5	0
10	10	10	10	10	10	10	10	10	0
50	50	50	27	27	27	50	50	25	25
100	85	85	42	42	42	85	79	39	40
500	403	403	157	157	136	403	358	120	238
1000	791	791	283	283	242	791	691	211	480
5000	3732	3732	1033	1033	833	3732	3102	692	2410
10000	5021	5021	1275	1275	1009	5021	4104	824	3280

All numbers are averaged of ten series of measurements.

Table F.5.4: WFS results for the hara.tc setting

hara.tc setting									
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	'200 OK'	'502 Bad Gateway'
1	1	1	1	1	1	1	1	1	0
5	5	5	5	5	5	5	5	5	0
10	10	10	10	10	10	10	10	10	0
50	50	50	26	26	26	50	50	26	24
100	84	84	41	41	41	84	77	38	39
500	393	393	146	146	130	393	338	116	222
1000	771	771	263	263	231	771	649	203	446
5000	3612	3612	943	943	790	3612	2854	662	2192
10000	5113	5113	1220	1220	1006	5113	3955	830	3125

All numbers are averaged of ten series of measurements.

F.5.2 Fingerprinting Series

Table F.5.5: Fingerprinting results for the LAN setting

LAN Setting												
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Requests	Requests received by FP Server	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	'200 OK'	'502 Bad Gateway'	Actually Sucessful Fingerprints
1	1	1	1	8	8	1	1	1	1	1	0	1
5	5	5	5	40	40	5	5	5	5	5	0	5
10	10	10	10	80	64	10	10	10	10	10	0	6
50	50	50	25	200	138	25	25	50	50	13	37	14
100	100	100	46	368	241	46	46	100	100	22	78	25
500	500	500	173	1384	775	173	173	500	500	64	436	81
1000	1000	1000	314	2512	1334	314	314	1000	1000	107	893	139
5000	5000	5000	1177	9416	3460	1177	1177	5000	5000	308	4692	362
10000	10000	10000	2138	17104	4830	2138	2138	10000	10000	513	9487	505

All numbers are averaged of ten series of measurements.

Table F.5.6: Fingerprinting results for the ‘CARLOS’ setting

‘CARLOS’ setting												
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Requests	Requests received by FP Server	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	‘200 OK’	‘502 Bad Gateway’	Actually Successful Fingerprints
1	1	1	1	8	8	1	1	1	1	1	0	1
5	5	5	5	40	40	5	5	5	5	5	0	5
10	10	10	10	80	64	10	10	10	10	10	0	6
50	26	26	25	200	138	25	24	26	26	24	2	14
100	48	48	47	376	246	47	46	48	48	46	2	25
500	183	183	178	1424	797	178	173	183	162	153	9	83
1000	335	335	325	2600	1381	325	315	335	292	275	17	144
5000	1277	1277	1229	9832	3613	1229	1183	1277	1061	983	78	378
10000	2333	2333	2241	17928	4803	2241	2152	2333	1906	1759	147	502

All numbers are averaged of ten series of measurements.

Table F.5.7: Fingerprinting results for the the-tester.de setting

the-tester.de setting												
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Requests	Requests received by FP Server	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	‘200 OK’	‘502 Bad Gateway’	Actually Sucessful Fingerprints
1	1	1	1	8	8	1	1	1	1	1	0	1
5	5	5	5	40	40	5	5	5	5	5	0	5
10	10	10	10	80	64	10	10	10	10	10	0	6
50	50	50	28	224	154	28	28	50	50	26	24	16
100	82	82	42	336	220	42	42	82	76	39	37	23
500	385	385	156	1248	699	156	136	385	342	120	222	73
1000	753	753	281	2248	1194	281	240	753	658	209	449	124
5000	3510	3510	1024	8192	3010	1024	826	3510	2917	686	2231	315
10000	4998	4998	1343	10744	3746	1343	1062	4998	4085	868	3217	392

All numbers are averaged of ten series of measurements.

Table F.5.8: Fingerprinting results for the hara.tc setting

hara.tc setting												
Parallel Client Requests	Requests received by INSANE	INSANE Requests	Requests received by BSCSM	BSCSM Requests	Requests received by FP Server	BSCSM Replies	Replies received by INSANE	INSANE Replies	Replies received by Client	'200 OK'	'502 Bad Gateway'	Actually Successful Fingerprints
1	1	1	1	8	8	1	1	1	1	1	0	1
5	5	5	5	40	40	5	5	5	5	5	0	5
10	10	10	10	80	64	10	10	10	10	10	0	6
50	50	50	27	216	149	27	27	50	50	27	23	15
100	85	85	42	336	220	42	42	85	78	39	39	23
500	405	405	154	1232	690	154	137	405	348	122	226	72
1000	796	796	278	2224	1181	278	244	796	670	214	456	123
5000	3762	3762	1014	8112	2981	1014	850	3762	2973	712	2261	312
10000	5033	5033	1243	9944	3467	1243	1025	5033	3893	845	3048	362

All numbers are averaged of ten series of measurements.

G USE CASES

In this chapter a rather comprehensive list of use cases is provided for the interested reader. However, the use cases are very abstract and are far from implementation details, as they are supposed to define the conceptual goals, rather than the implementation goals for the crowdsourcing extension of the MapBiquitous project. In the following the terms *client* and *device* are considered to be synonymous.

G.1 CLIENT→INSANE COMMUNICATION

G.1.1 Setter

Registration of a User

A user, may they want to participate in the crowdsourcing process for the first time or have they participated before and cancelled their participation, decides to participate in the crowdsourcing process. In order to participate, a submitter-ID is required, which is automatically issued as soon as the user uses a client and contacts an INSANE.

Should the INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

The INSANE responsible for the DHT area the username is associated with registers the user by having been sent a username, a password, a device identification (client-ID) as well as some device specifications and stores these information. The client-ID is used to create a corresponding client-account at the same time. Should the user-account (or a 'similar' one) already exist, the use case 'registration of a device' applies, otherwise a distribution-wide unique user-ID is calculated and a corresponding submission-ID generated. The user is notified of the creation by returning positive feedback, otherwise negative feedback is returned.

Registration of a Device

Having a user-account on an INSANE responsible for the DHT area the username is associated, the user needs to register their device(s) in order to use the system. For this, the same server interface that is also used for the 'registration of a user' use case is used.

Should the INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

There are several conceivable scenarios:

- The provided client-ID is not known to the INSANE. – The INSANE then creates a corresponding client profile and associates it with the user-account. Afterwards, positive feedback is returned to the user.
- The provided client-ID is already registered with a client profile that is associated with the user-account of the contacting user. – The INSANE returns a negative feedback to the user.
- The provided client-ID is already registered with a client profile that is not associated with the user-account of the contacting user. – The INSANE then returns a denial of registration to the user.

Submission of Crowdsourcing Data

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature of the submission are validated. Should validation fail, the user is sent a denial of processing feedback, else the submission is stored on the INSANE and forwarded to the building server affected by the submission. The forward is stripped down to the actual crowdsourcing data and then amended with the submitter-ID of the user and then signed by the INSANE. Iff the building server accepts the submission, the INSANE returns a positive feedback to the user, otherwise negative feedback is returned.

Deletion of one Submission of a User

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature of the submission are validated. Should validation fail, the user is sent a denial of processing feedback, else the provided submission-ID is

cross-referenced against the submission database on the INSANE. Should no submission be associated with the submission-ID the user is returned negative feedback, otherwise the building server affected by the submission is contacted and deletion of the submission on the building server is requested. Iff the building server accepts the deletion of the submission, the INSANE deletes the corresponding submission data from its database and returns a positive feedback to the user, otherwise negative feedback is returned.

Deletion of all Submissions of a User

Deletion of all submissions of a user is archived by multiple application of the 'deletion of one submission of a user' use case until no submission associated with the user remains in the database of the INSANE.

Deletion of a User

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature are validated. Should validation fail, the user is sent a denial of processing feedback, else all submissions of the user a deleted as described in the 'deletion of all submissions of a user' use case. Failed deletions are ignored. Afterwards all device-profiles associated with the user are deleted (also ignoring errors) as described in the 'deletion of a device' use case; then, the user-profile is deleted from the database. A positive feedback is returned to the user.

Deletion of a Device

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature are validated. Should validation fail, the user is sent a denial of processing feedback, else the provided client-ID is cross-referenced against the database-table of devices on the INSANE. Should no device be associated with the client-ID the user is returned negative feedback, otherwise the corresponding device-profile is deleted and a positive feedback is returned to the user.

Correction of a Submission

Correction of a submission is achieved by application of the 'deletion of one submission of a user' use case followed by application of the 'submission of crowdsourcing data' use case with the corresponding corrected data.

G.1.2 Getter

Retrieval of the INSANE responsible for a User

A user may request information on their responsible INSANE by sending their username to any INSANE they know. The contacted INSANE determines the geographic location based on the user's IP-address. Iff the INSANE does not know the geographic region, the region of the contacted INSANE is assumed. Within the region, the INSANE looks up the responsible INSANE based on the hashed username and the distributed hash table of INSANEs. Should the responsible INSANE not be known, the INSANE returns information on an INSANE closer to the responsible INSANE with respect to the distributed hash table.

Comment

> *It is important to have the geographic region – this should be the country for obvious reasons – checked first, as the contacted INSANE may be responsible for the user in the region the contacted INSANE is situated at. However, the user may reside outside of the INSANE's region, in which case the INSANE would falsely declare itself responsible.*

Retrieval of a List of INSANEs

A user contacts an INSANE. In return, the contacted INSANE replies with all INSANEs known to the contacted INSANE.

Anonymous Retrieval of Data from a Building Server

Any getter interface on building servers available for direct client access can be proxy-accessed via an INSANE. The same information required to use the interface on the building server has to be provided to the INSANE's interface. The INSANE forwards the user's request to the building server, receives the reply from the building server and forwards the reply to the user. No crowdsourcing related information is exchanged between INSANE and building server and the user's identity (especially the IP-address) is concealed from the building server.

User-specific Retrieval of Data from a Building Server

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature of the submission are validated. Should validation fail, the 'anonymous retrieval of data from a building server' use case applies.

Any getter interface on building servers available for direct client access can be proxy-accessed via an INSANE. The same information required to use the interface on the building server has to be provided to the INSANE's interface. Additionally, the user must provide their credentials. The

INSANE amends the user's request with the corresponding submitter-ID and forwards it to the building server. On basis of the provided submitter-ID the building server generates a modified reply containing crowdsourced information of the user. The INSANE receives this personalised reply from the building server and forwards the reply to the user. The user's real identity (especially the IP-address) is concealed from the building server; however, the user's crowdsourcing identity (based on the submitter-ID) is disclosed.

Retrieval of all Submissions of a User

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature of the submission are validated. Should validation fail, the user is sent a denial of processing feedback, else the contacted INSANE generates a list of all submissions of the user based on its database entries. Should no submissions be stored on the INSANE, a negative feedback is returned to the user, otherwise a positive feedback.

Retrieval of the Capability List of a User

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature of the submission are validated. Should validation fail, the user is sent a denial of processing feedback, else the contacted INSANE generates a list of all capabilities of the user based on its database entries. Should no capabilities be stored on the INSANE, a negative feedback is returned to the user, otherwise a positive feedback.

G.1.3 Retrieval of a User's own Submitter-ID

The user is required to have a valid user-profile on a responsible INSANE.

Should the contacted INSANE not be responsible for the DHT area the username is associated with, the INSANE supports lookup of the responsible INSANE or an INSANE that may be able to lookup or know the responsible INSANE. Lookup is continued until the responsible INSANE is contacted.

Should there be no valid user-profile on the responsible INSANE, a negative feedback is returned to the user, otherwise the password and signature of the request are validated. Should validation fail, the user is sent a denial of processing feedback, else the contacted INSANE returns the submitter-ID corresponding to the contacting user.

G.2 INSANE→BUILDING SERVER COMMUNICATION

G.2.1 Setter

Comment

The use cases of this subsection assume sufficient rights of the user on the building server. Should the user fail to have the necessary access privileges, the building server returns a denial of processing response to the INSANE.

Submission of crowdsourced Data

This use case can only apply iff a user has requested submission of crowdsourcing data on an INSANE.

After receiving a request for submission of crowdsourced data by a user, an INSANE contacts the building server responsible for the building the submission affects. The INSANE is required to be known to the building server. Should the contacted building server not know the contacting INSANE, the INSANE's information is looked up via the directory service. Should no information be available, the building server denies the submission with a negative feedback. This denial is forwarded as negative feedback to the user the request for submission originated at.

Should the INSANE be known to the building server, the signature of the request of the INSANE is validated. Should validation fail, the INSANE is sent a denial of processing feedback, else the submission's submission-ID is checked. Should a submission with the same submission-ID already exist in the building server's database, the INSANE is sent a denial of processing feedback, otherwise the submission is stored on the building server for further processing and the INSANE is replied to with a positive feedback. Should the submitter-ID of the submission not be known to the building server, a new submitter account is created, otherwise the submission is amended to the list of submissions of the (known) submitter.

Deletion of crowdsourced Data

This use case can only apply iff a user has requested deletion of crowdsourcing data or deletion of the user account or device on an INSANE.

After receiving a request for deletion of a submission by a user, an INSANE contacts the building server responsible for the building the submission deletion affects. The INSANE is required to be known to the building server. Should the contacted building server not know the contacting INSANE, the INSANE's information is looked up via the directory service. Should no information be available, the building server denies the selection of the submission with a negative feedback. This denial is forwarded as negative feedback to the user the request for submission originated at.

Should the INSANE be known to the building server, the signature of the request of the INSANE is validated. Should validation fail, the INSANE is sent a denial of processing feedback, else the submission's submission-ID and submitter-ID are checked. Should no submission with the provided IDs exist in the building server's database, the INSANE is sent a negative feedback, otherwise the submission is deleted on the building server and the INSANE is replied to with a positive feedback.

Correction of crowdsourced Data

This use case should not be considered for implementation as the INSANE's side of this use case envisages correction of submissions by deleting the old submission and submitting a new one.

G.2.2 Getter

Retrieval of sifted Data

This is the default use case for getter-access. It is valid for anonymous access as well as access with false credentials.

This use case can only apply iff a user has requested retrieval of data on an INSANE.

After receiving a request for retrieval of data by a user, an INSANE contacts the building server responsible for the building the retrieval affects. The INSANE is not required to be known to the building server.

The building server returns the requested data on basis of the most recently sifted data-set within its database.

Comment

> *Theoretically, this use case can also apply for user access this building server directly; however, the intent is to allow anonymous access, which is reduced to absurdity should the user contact the building server directly.*

Retrieval of a User-specific intermediate Crowdsourcing Data

This use case can only apply iff a user has requested retrieval of data on an INSANE.

After receiving a request for retrieval of data by a user, an INSANE contacts the building server responsible for the building the retrieval affects. The INSANE is required to be known to the building server. Should the contacted building server not know the contacting INSANE, the INSANE's information is looked up via the directory service. Should no information be available, the 'retrieval of sifted data' use case applies.

Should the INSANE be known to the building server, the signature of the request of the INSANE is validated. Should validation fail, the 'retrieval of sifted data' use case applies, else the submitter-ID is checked. Should submissions with the same submitter-ID exist in the building server's database, the INSANE these submissions are processed into the sifted data. The modified data is replied to the INSANE. Should no such submissions exist, the unmodified sifted data is returned.

G.3 INSANE←BUILDING SERVER COMMUNICATION

G.3.1 Setter

Registration of a User's Access Rights

Iff a user has not participated in the crowdsourcing, yet, when conducting their first write-access via the INSANE, the INSANE will store the username, the used password, the client-ID, etc., and generate a submitter-ID as well as issue a general privilege pointer collection with no privilege pointers[156] and default time of validity, which is stored alongside the user data. A copy of the privilege pointer collection can then be sent to the user's client.
Then, the following may apply:

- A user has not participated in the crowdsourcing, yet. Before actually using the system, they would request privileges on a building server. The maintainer of the building server – basically an administrator – would then
  - deny the privileges (end of use case), or
  - grant the privileges, leading to the issuing of a building server initiated privilege pointer. For this, the maintainer of the building server would request the user to submit their submitter-ID and create a corresponding entry in the ACL. The list of INSANEs this ACL entry has been shared with is empty.
    As soon as the user first contacts the building server via an INSANE, the building server checks whether the provided submitter-ID has a corresponding entry in the ACL and if the contacting INSANE is in the list of INSANEs the ACL entry has been shared with. As the contacting INSANE cannot be found, the privilege pointer is pushed to the INSANE alongside the submitter-ID, and the INSANE is added to the list of INSANEs the ACL entry has been shared with. Receiving this push, the INSANE inserts the privilege pointer into the user's privilege pointer collection. The new privilege pointer collection can then be sent to the user's client. (end of use case)
- A user has already participated in the crowdsourcing. When accessing the INSANE their privilege pointer collection – or parts of it – are deemed expired. Any privilege pointer stored in the old privilege pointer collection which has not expired, is copied into a newly issued privilege pointer collection, while any expired privilege pointer is not copied and drered. The new privilege pointer collection can then be sent to the user's client. (end of use case)
- A user has already participated in the crowdsourcing. They request privileges on a building server; hence, the maintainer of the building server would then
  - deny the privileges (end of use case), or
  - grant the privileges, leading to the issuing of a building server initiated privilege pointer. For this, the maintainer of the building server would request the user to submit their submitter-ID and create a corresponding entry in the ACL. The list of INSANEs this ACL entry has been shared with is empty.
    As soon as the user contacts the building server via an INSANE the next time, the building server checks whether the provided submitter-ID has a corresponding entry in the ACL with the contacting INSANE not being in the list of INSANEs the ACL entry has been shared with. As the contacting INSANE cannot be found in the list, the privilege pointer is
    pushed to the INSANE alongside the submitter-ID, and the INSANE is added to the list of INSANEs the ACL entry has been shared with. Receiving this push, the INSANE inserts the privilege pointer into the user's privilege pointer collection. The new privilege pointer collection can then be sent to the user's client. (end of use case)

[156]This empty privilege pointer collection can be used by any building server to distribute a privilege pointer to the server's ACL information of the corresponding user.

Revocation of a User's Access Rights

The maintainer of a building server wants to revoke a user's access rights. After identifying all ACL entries affected by this revocation, the building server sends a revocation command including the submitter-ID and the privilege pointer ID(s) to all INSANEs in the list of INSANEs the identified ACL entries had been shared with. Each contacted INSANE must remove the corresponding privilege pointer from the user's privilege pointer collection.

Request for Blocking of a User

For this use case the submitter-ID of the user to be blocked must be known to the maintainer of a building server[157].

The maintainer of a building server wants to block a user in order to prevent accesses from that user. Corresponding to the 'registration of a user's access rights' use case, the building server creates denying ACL entries for the user. However, the privilege pointers are marked as blocking privilege pointers and rather than only storing the privilege pointers into the user's privilege pointer collections, all INSANEs storing the blocking privilege pointer for the user must actively deny all access to the building server.

[157]This should be the case as a maintainer is only supposed to block users after identifying them as spammers, scammers, etc.

H INTERFACE DEFINITIONS

All interfaces described within this section, regardless of the server they are hosted on (INSANE, BSCSM or directory service), are only accessible by HTTP POST requests. This ensures that the request variables are part of the HTTP header of the requests and are encrypted alongside the rest of the HTTP packets[158]. Further, interfaces involving the transmission, storage or access of timestamps use the ISO 8601 'Zulu' format[159], which bases all timestamps on the universal coordinated time (UTC)[160], e.g. '2012-10-05T16:48:13Z' is the 'Zulu' representation of the point in time at 5 October 2012, 18 hours, 48 minutes and 13 seconds central European summer time (CEST).

Comment

> *The client→INSANE interfaces following in this chapter are designed to act jointly with the MapBiquitous client conceived by Gerd Bombach. Therefore, these interface definitions are referenced to by [Bom12], but they should not be considered sole work of the author of this thesis. In lieu thereof, they should be considered a team effort of Gerd Bombach and the author of this thesis.*

H.1 CLIENT→INSANE

H.1.1 INSANE-internal Setter

Registration of a User and/or Device

Registers a new user and a new device or adds a device to an existing user.

- Call:
  - `method`=*registerUser*`&userName`=*USERNAME*`&password`=*PASSWORD* `&clientID`=*SHA-256*`&model`=*MODEL-NAME*`&publicKey`=*PUBLIC-KEY* `&signature`=*SIGNATURE*

[158] GET requests transfer their variables within the access URL.

[159] '*Year*-*Month*-*Day*`T`*Hour*`:`*Minute*`:`*Second*`Z`'

[160] UTC is comparable to the Greenwich Mean Time (GMT) as it seems to be the same during standard time, but it has no daylight saving time. Additionally, GMT may vary up to 0.9 seconds from UTC as GMT is a national standard, while UTC is an international standard.

- HTTP Replies (header | body):
  - 201 Created |
  - 303 See Other |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
- Expectations:
  - after encountering error 303 `getMyINSANE` should be called to reach the responsible INSANE
  - a 400 error suggests an incorrect file format, incompatible interface versions, already existing user name or similar problems; this problem must be solved by the client
  - a 403 error suggests an invalid signature; the client should resubmit its request using a valid signature

Change a User's Password

Changes a registered user's password.

- Call:
  - `method`=*changePassword*`&userName`=*USERNAME*`&oldPassword`=*PASSWORD* `&newPassword`=*PASSWORD*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK |
  - 303 See Other |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 404 Not Found |
- Expectations:
  - after encountering error 303 `getMyINSANE` should be called to reach the responsible INSANE
  - a 400 error suggests an incorrect file format, incompatible interface versions, already existing user name or similar problems; this problem must be solved by the client
  - a 403 error suggests an invalid signature or password; the client should resubmit its request using valid signature and password
  - a 404 error suggests that this user is not known

H.1.2 INSANE-internal Getter

Determine the INSANE's Hash Region in the DHT

Reports the DHT region managed by the current INSANE.

- Call:
  - `method`=*getDHTArea*
- HTTP Replies (header | body):
  - 200 OK | DHT Region
- Expectations:
  - none

Identify neighbouring INSANEs

Reports a list of all INSANEs known to the current INSANE.

- Call:
  - `method`=*getDHTNeighbors*
- HTTP Replies (header | body):
  - 200 OK | List of all known INSANEs
- Expectations:
  - none

Determine responsible INSANE

Establishes the current INSANE's responsibilities; if this INSANE is not responsible for this user it either, if known, reports the responsible INSANE or, if not known, another INSANE possibly capable of helping out; this is a mixture of DNS and DHT.

- Call:
  - `method`=*getMyINSANE*`&username`=*USERNAME*
- HTTP Replies (header | body):
  - 302 Found | The INSANE able to respond or relegate
  - 409 Conflict | Random INSANE able to respond or relegate
  - 400 Bad Request | Error Message
  - 404 Not Found |
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - as a 404 error can only occur due to an error in the DHT distribution the request should be resubmitted after a short waiting period; possibly the DHT is being reconstructed and the necessary DHT region has not been reorganized

Get Public Key

Reports the current INSANE's public key.

- Call:
  - `method`=*getPublicKey*
- HTTP Replies (header | body):
  - 200 OK | Public Key
- Expectations:
  - none

Fetch a User's Submissions

Reports all submissions by the submitted user.

- Call:
  - `method`=*getMySubmissions*`&password`=*PASSWORD*`&clientID`=*SHA-256* `&signature`=*SIGNATURE*

- HTTP Replies (header | body):
  - 200 OK / timestamp / signature* | [submissionID:Submission-JSON, ...]
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 404 Not Found |

  *: signature is calculated using '`result`=*packet-body*`&timestamp`=*2012-10-02T00:39:41Z*'
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature
    - that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
  - a 404 error suggests that there is no existing submission from this device
  - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method

Fetch a User's WLAN Position Correction Submissions

Reports all WLAN position correction submissions by the submitted user.

- Call:
  - `method`=*getMyWLANFingerprintingPositionCorrectionSubmissions* `&password`=*PASSWORD*`&clientID`=*SHA-256*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK / timestamp / signature* | [submissionID:Submission-JSON, ...]
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 404 Not Found |

  *: signature is calculated using '`result`=*packet-body*`&timestamp`=*2012-10-02T00:39:41Z*'
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature
    - that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
  - a 404 error suggests that there is no existing submission from this device
  - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method

Fetch a User's GSM Fingerprinting Submissions

Reports all GSM fingerprinting submissions by the submitted user.

- Call:
  - `method`=*getMyGSMFingerprintSubmissions*`&password`=*PASSWORD* `&clientID`=*SHA-256*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK / timestamp / signature* | [submissionID:Submission-JSON, ...]
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 404 Not Found |

*: signature is calculated using '`result`=*packet-body*`&timestamp`=*2012-10-02T00:39:41Z*'

- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature
    - that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
  - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method

Fetch a User's WLAN Fingerprinting Submissions

Reports all WLAN fingerprinting submissions by the submitted user.

- Call:
  - `method`=*getMyWLANFingerprintSubmissions*`&password`=*PASSWORD* `&clientID`=*SHA-256*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK / timestamp / signature* | [submissionID : Submission-JSON, ...]
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 404 Not Found |

*: signature is calculated using '`result`=*packet-body*`&timestamp`=*2012-10-02T00:39:41Z*'

- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature
    - that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
  - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method

H.1.3 Passed-through to BSCSM Setter

Delete a User or Device

Deletes one device of a user or the user including all their devices (if the user does not want to participate in the crowdsourcing process anymore). All submissions by this user or device are deleted.

- Call:
  - `method`=*deleteUser*`&username`=*USERNAME*`&password`=*PASSWORD* `&clientID`=*SHA-256*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK |
  - 303 See Other |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
- Expectations:

  - ClientID is optional, if given only this client is deleted, otherwise the user and all clients are deleted
  - after encountering error 303 `getMyINSANE` should be called to reach the responsible INSANE
  - a 400 error suggests an incorrect file format, incompatible interface versions, already existing user name or similar problems; this problem must be solved by the client
  - a 403 error suggests an invalid signature or password; the client should resubmit its request using valid signature and password

Manual WLAN Position Correction (Aware Direct CS)

Corrects WLAN fingerprint data based on fingerprints previously submitted.

- Call:
  - `method`=*correctWLANFingerprintingPosition*`&password`=*PASSWORD* `&clientID`=*SHA-256*`&BS`=*BUILDINGSERVER-ID*`&oldPositon`=*POSITION-JSON* `&newPosition`=*POSITION-JSON*`&fingerprintData`=*FINGERPRINT-JSON* `&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 404 Not Found |
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
    - a 403 error suggests:
      an invalid signature; the client should resubmit its request using a valid signature
      * that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
  - a 404 error suggests that the old position does not match the position reported by the fingerprinting server; the INSANE should request the position from the building server
  - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method
- Annotations:
  - Structure of a *POSITION-JSON*:
    * ```
      {
          "building" :  "tud_inf:TUD_INF",
          "provider" :  "FINGERPRINT",
          "latitude" :  51.0256759542,
          "wt_sec" :  0,
          "longitude" :  13.7231337647,
          "heading" :  0,
          "altitude" :  0,
          "timestamp" :  1348587573,
          "accuracy" :  0,
          "level" :  3
      }
      ```
      ```

Submit GSM Data (Unaware Direct CS, Unaware Indirect CS)

Submits continuously collected GSM data to generate a new layer on the building server.

- Call:
  - `method`=*createGSMFingerprint*`&password`=*PASSWORD*`&clientID`=*SHA-256* `&BS`=*BUILDINGSERVER-ID*`&gsmData`=*GSM-JSON* `&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature
    - that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
  - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method
- Annotations:
  - Structure of a *GSM-JSON*:

```
{
      "GSMmeasurment" :  {
          "Location" :  "51.0257241256442;13.722872620470596",
          "Building" :  "INF",
          "Floor" :  "3",
          "CellID" :  "29773",
          "Provider" :  "Vodafone",
          "Strenght" :  "-63",
          "Time" :  timestamp
          "Fingerprint" :  {...}
      },
      "GSMmeasurment" :  {...}
  }
```

Upload WLAN Fingerprint (Unaware Indirect CS)

Submit a new WLAN fingerprint, multiple measurements are possible.

- Call:
  - `method`=*createWLANFingerprint*`&password`=*PASSWORD*`&clientID`=*SHA-256* `&BS`=*BUILDINGSERVER-ID*`&position`=*POSITION-JSON*`&samples`=*SAMPLES-JSON* `&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature

        * that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
    - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method
- Annotations:
    - Structure of a *POSITION-JSON*:

```
{
      "building" :  "tud_inf:TUD_INF",
      "latitude" :  51.02567595421748,
      "wt_sec" :  0.0,
      "longitude" :  13.723133764723404,
      "heading" :  0.0,
      "altitude" :  0.0,
      "timestamp" :  0,
      "accuracy" :  0.0,
      "level" :  3
  }
```

    - Structure of a *SAMPLES-JSON*:

```
[
      {
          "ap_mac" :  "08:17:35:33:59:00",
          "y" :  0.0,
          "x" :  0.0,
          "latitude" :  0.0,
          "longitude" :  0.0,
          "ap_signal" :  -68,
          "id" :  0,
          "floor" :  0
      },
      {"ap_mac" :  ...}
  ]
```

Delete Submission

Deletes a specific submission.

- Call:
    - `method`=*deleteSubmission*`&password`=*PASSWORD*`&clientID`=*SHA-256* `&submissionID`=*ISO-8601-TIMESTAMP:SHA-256*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
    - 200 OK |
    - 400 Bad Request | Error Message
    - 403 Forbidden |
    - 404 Not Found |
    - 502 Bad Gateway | Error Message
- Expectations:
    - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
    - a 403 error suggests:
        * an invalid signature; the client should resubmit its request using a valid signature
        * that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
    - a 502 error suggests:
        * an invalid signature within the building server's response; the client is to decide whether to repeat the request
        * an error within the building server, e.g. 500
    - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method

H.1.4 Passed-through to BSCSM Getter

Fetch WFS Data from a Building Server

Requests WFS data from a building server using the INSANE. If user access information is provided the results include the user's crowdsourcing submissions, otherwise the WFS request is anonymous and only public information is returned.

- Call:
  - `method`=*getWFSDataFromBS*`&password`=*PASSWORD*`&clientID`=*SHA-256* `&BS`=*BUILDINGSERVER-ID*`&SERVICE`=*SERVICE*`&VERSION`=*VERSION* `&REQUEST`=*REQUEST*`&TYPENAME`=*TYPENAME*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK / timestamp / signature* | Computed building server response
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 502 Bad Gateway | Error Message

  *: signature is calculated using '`result`=*packet-body*`&timestamp`=*2012-10-02T00:39:41Z*'
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature
    - that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
  - a 502 error suggests:
    - an invalid signature within the building server's response; the client is to decide whether to repeat the request
    - an error within the building server, e.g. 500
  - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method

Fetch Fingerprinting Position from a Building Server

Requests fingerprinting data from a building server using the INSANE. If user access information is provided the results include the user's crowdsourcing submissions, otherwise the fingerprinting request is anonymous and only public information is returned.

- Call:
  - `method`=*getWLANFingerprintingPositionFromBS*`&password`=*PASSWORD* `&clientID`=*SHA-256*`&BS`=*BUILDINGSERVER-ID*`&accesspoints`=*AP-JSON* `&prev_position`=*PP-JSON*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK / timestamp / signature* | Computed building server response
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 502 Bad Gateway | Error Message

  *: signature is calculated using '`result`=*packet-body*`&timestamp`=*2012-10-02T00:39:41Z*'
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client
  - a 403 error suggests:
    - an invalid signature; the client should resubmit its request using a valid signature

        * that the client device is not known to the current INSANE; most likely the current INSANE is not responsible for the user's DHT region
    - a 502 error suggests:
        * an invalid signature within the building server's response; the client is to decide whether to repeat the request
        * an error within the building server, e.g. 500
    - to avoid the 403 error a final `getMyINSANE` request should be completed before calling this method
- Annotations:
    - Structure of an *AP-JSON*:
        * [
            {
                "ssid" : "08:17:35:33:59:0F",
                "signal_strength" : -67
            },
            {
                "ssid" : "08:17:35:33:59:01",
                "signal_strength" : -67
            },
            ...
          ]
    - Structure of a *PP-JSON*:
        * {
            "building" : "tud_inf:TUD_INF",
            "provider" : "FINGERPRINT",
            "latitude" : 51.0256759542,
            "wt_sec" : 0,
            "longitude" : 13.7231337647,
            "heading" : 0,
            "altitude" : 0,
            "timestamp" : 1348587573,
            "accuracy" : 0,
            "level" : 3
          }

H.2 INSANE→BSCSM

H.2.1 BSCSM-internal Setter

Submit GSM Data (Unaware Direct CS, Unaware Indirect CS)

Submits continuously collected GSM data to generate a new layer on the building server.

- Call:
    - `method`=*createGSMFingerprint*`&submissionID`=*ISO-8601-TIMESTAMP:SHA-256* `&insaneID`=*INSANE-ID*`&submitterID`=*SHA-256*`&model`=*MODEL* `&gsmData`=*GSM-JSON*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
    - 202 Accepted |
    - 400 Bad Request | Error Message
    - 403 Forbidden |
    - 502 Bad Gateway |

- Expectations:
  - a 400 error suggests an incorrect file format, incompatible interface versions or the submissionID's SHA-256 Hash not corresponding to the submission's content
  - a 403 error suggests:
    * an invalid signature; the INSANE should resubmit its request using a valid signature
    * the calling INSANE is not known
    * the submitter account using this submitterID does not have the rights necessary to upload GSM fingerprinting submissions
    * there already exists a submission using this submissionID
  - a 502 error suggests failed internal communication between BSCM and fingerprinting module
  - As this method supports INSANE autodiscover using the directory service, information can be retrieved by unknown INSANEs. Nevertheless the communication protocol stipulates that an INSANE is to check its registration status using `getINSANERegistrationState` before calling this method.
    If `getINSANERegistrationState` returns 404 the calling INSANE should register itself using `registerAtINSANE`. Only iff a 200 is received may the INSANE call `createGSMFingerprint`.
- Annotations:
  - Structure of a GSM-JSON: analogous to `createGSMFingerprint` at an INSANE

Submit WLAN fingerprint (Unaware Indirect CS)

Submit a new WLAN fingerprint, multiple measurements are possible.

- Call:
  - `method`=*createWLANFingerprint*`&submissionID`=*ISO-8601-TIMESTAMP:SHA-256* `&insaneID`=*INSANE-ID*`&submitterID`=*SHA-256*`&model`=*MODEL* `&position`=*POSITION-JSON*`&samples`=*SAMPLES-JSON* `&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 202 Accepted |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 502 Bad Gateway |
- Expectations:
  - a 400 error suggests an incorrect file format, incompatible interface versions or the submissionID's SHA-256 Hash not corresponding to the submission's content
  - a 403 error suggests:
    * an invalid signature; the INSANE should resubmit its request using a valid signature
    * the calling INSANE is not known
    * the submitter account using this submitterID does not have the rights necessary to upload WLAN fingerprinting submissions
    * there already exists a submission using this submissionID
  - a 502 error suggests failed internal communication between BSCM and fingerprinting module
  - As this method supports INSANE autodiscover using the directory service, information can be retrieved by unknown INSANEs. Nevertheless the communication protocol stipulates that an INSANE is to check its registration status using `getINSANERegistrationState` before calling this method.
    If `getINSANERegistrationState` returns 404 the calling INSANE should register itself using `registerAtINSANE`. Only if a 200 is received may the INSANE call `createWLANFingerprint`.
- Annotations:
  - Structure of the *POSITION-JSON* and *SAMPLES-JSON* is analogous to `createWLANFingerprint` at an INSANE

Delete Submission

Deletes a submission by Submission-ID.
(this method must be invoked by an INSANE)

- Call:
  - `method`=*deleteSubmission*`&submissionID`=*ISO-8601-TIMESTAMP:SHA-256*
    `&insaneID`=*INSANE-ID*`&submitterID`=*SHA-256*
    `&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK |
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 404 Not Found |
  - 502 Bad Gateway | Error Message
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the INSANE
  - a 403 error suggests:
    - an invalid signature; the INSANE should resubmit its request using a valid signature
    - the current building server does not know the calling INSANE
    - the submitter account using this submitterID does not have the rights necessary to upload WLAN fingerprinting submissions
  - a 404 error suggests:
    - there is no known submitter using this Submitter-ID
    - there is no submission by this Submission-ID
  - a 502 error suggests failed internal communication between BSCM and fingerprinting module
  - As this method supports INSANE autodiscover using the directory service, information can be retrieved by unknown INSANEs. Nevertheless the communication protocol stipulates that an INSANE is to check its registration status using `getINSANERegistrationState` before calling this method.
    If `getINSANERegistrationState` returns 404 the calling INSANE should register itself using `registerAtINSANE`. Only if a 200 is received may the INSANE call `deleteSubmission`.

H.2.2 BSCSM-internal Getter

Determine Position based on Fingerprinting Information

Computes the most likely position using the submitted WLAN fingerprint.

- Call:
  - `method`=*getWLANFingerprintingPosition*`&accesspoints`=*FP-JSON*
    `&prev_position`=*Position-JSON*`&insaneID`=*INSANE-ID*`&submitterID`=*SHA-256*
    `&timestamp`=*TIMESTAMP*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK | XML representation including the position
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 502 Bad Gateway |

- Expectations:
  - Three access types are supported:
    - *INSANE-ID and submitterID empty*:
      Direct access by a client; no Anonymity; fingerprinting information is retrieved without alterations
    - *INSANE-ID set, submitterID empty*:
      Anonymised access using an INSANE; fingerprinting information is retrieved without alterations
    - *INSANE-ID and submitterID set*:
      Crowdsourcing-based access; fingerprinting information is retrieved including submissions by submitterID
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the client/INSANE
  - a 403 error suggests an invalid signature; the INSANE should resubmit its request using a valid signature
  - a 502 error suggests failed internal communication between BSCM and fingerprinting module
  - As this method supports INSANE autodiscover using the directory service, information can be retrieved by unknown INSANEs. Nevertheless the communication protocol stipulates that an INSANE is to check its registration status using `getINSANERegistrationState` before calling this method.
    If `getINSANERegistrationState` returns 404 the calling INSANE should register itself using `registerAtINSANE`. Only if a 200 is received may the INSANE call `getWLANFingerprintingPosition`.

Fetch WFS Information

Reports WFS information retrieved from the geoserver module, either anonymously or crowdsourcing-based.

- Call:
  - `method`=*getWFS*`&SERVICE`=*SERVICE*`&VERSION`=*VERSION*`&REQUEST`=*REQUEST*
    `&TYPENAME`=*TYPENAME*`&insaneID`=*INSANE-ID*`&submitterID`=*SHA-256*
    `&timestamp`=*TIMESTAMP*`&signature`=*SIGNATURE*
- HTTP Replies (header | body):
  - 200 OK | XML representation including the position
  - 400 Bad Request | Error Message
  - 403 Forbidden |
  - 502 Bad Gateway |
- Expectations:
  - Two access types are supported:
    - INSANE-ID set, submitterID empty:
      Anonymised access using the INSANE; WFS information is retrieved without alterations
    - INSANE-ID and submitterID set:
      Crowdsourcing-based access; WFS information is retrieved including submissions by submitterID
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the INSANE
  - a 403 error suggests an invalid signature; the INSANE should resubmit its request using a valid signature
  - a 502 error suggests failed internal communication between BSCM and fingerprinting module
  - As this method supports INSANE autodiscover using the directory service, information can be retrieved by unknown INSANEs. Nevertheless the communication protocol stipulates that an INSANE is to check its registration status using `getINSANERegistrationState` before calling this method.
    If `getINSANERegistrationState` returns 404 the calling INSANE should register itself using `registerAtINSANE`. Only if a 200 is received may the INSANE call `getWFS`.

Determine Registration Status of an INSANE

Checks whether an INSANE is already registered with the building server.

- Call:
  - `method`=*getINSANERegistrationState*`&insaneID`=*INSANE-ID*
- HTTP Replies (header | body):
  - 302 Found |
  - 400 Bad Request | Error Message
  - 404 Not Found |
- Expectations:
  - a 400 error suggests an incorrect file format, incompatible interface versions, already existing user name or similar problems; this problem must be solved by the INSANE

Get Public Key

Reports the public key of the current INSANE.

- Call:
  - `method`=*getPublicKey*
- HTTP Replies (header | body):
  - 200 OK | Public Key
- Expectations:
  - none

H.2.3 INSANE-endorsed BSCSM-internal/external Setter

Registration at an INSANE

Initiates the information exchange between an INSANE and a BSCSM. The BSCSM should register itself at the calling INSANE, while the INSANE is registered at the building server.

- Call:
  - `method`=*registerAtINSANE*`&insaneName`=*INSANE-ID*
- HTTP Replies (header | body):
  - 201 Created |
  - 400 Bad Request | Error Message
  - 404 Not Found |
  - 424 Failed Dependency |
- Expectations:
  - a 400 error suggests an incorrect file format or incompatible interface versions; this problem must be solved by the INSANE
  - a 424 error suggests one of two cases:
    - the directory service could not be reached
    - the INSANE was found using the directory service but does not reply or encountered an error when calling the method `registerBuildingServer`

H.3 INSANE←BSCSM

H.3.1 INSANE-internal Setter

Register a Building Server

Registers a building server at the INSANE.

- Call:
  - `method`=*registerBuildingServer*`&bsName`=*BUILDINGSERVER-ID*
- HTTP Replies (header | body):
  - 201 Created |
  - 400 Bad Request | Error Message
  - 404 Not Found |
  - 502 Bad Gateway |
- Expectations:
  - a 400 error suggests an incorrect file format, incompatible interface versions, already existing user name or similar problems; this problem must be solved by the building server
  - a 404 error suggests that the directory service could not find a building server by this name
  - a 502 error suggests an error within the directory service or a failed call of `getPublicKey` following a successful directory request

H.4 ∗ →DIRECTORY SERVICE

H.4.1 Directory Service Getter

Search for Building Servers by Geo-Location (Position Box)

Searches for building servers within a given position box, defined by minimum and maximum longitude as well as minimum and maximum latitude.

- Call:
  - `minLat`=*WGS84Lat*`&maxLat`=*WGS84Lat*`&minLon`=*WGS84Lon*`&maxLon`=*WGS84Lon*
- HTTP Replies (header | body):
  - 200 OK | XML representation of the building server information
  - 400 Bad Request | HTML resource
  - 404 Not Found |
- Expectations:
  - a 400 error suggests a missing variable; this problem must be solved by the requester

Search for a Building Server by Name

Searches for a building server by its name. This method should return none or exactly one building server, otherwise something is wrong.

- Call:
  - `name`=*NAME*
- HTTP Replies (header | body):
  - 200 OK | XML representation of the building server information
  - 400 Bad Request | HTML resource
  - 404 Not Found |
- Expectations:
  - a 400 error suggests a missing variable; this problem must be solved by the requester

Search for INSANEs by Geo-Location (RFC 920 Country Code)

Searches for INSANEs within a geo-location given by a RFC 920 country code, e.g. 'DE'.

- Call:
  - `region`=*RFC920-CODE*
- HTTP Replies (header | body):
  - 200 OK | XML representation of the INSANE information
  - 400 Bad Request | HTML resource
  - 404 Not Found |
- Expectations:
  - a 400 error suggests a missing variable; this problem must be solved by the requester

Search for an INSANE by Name

Searches for an INSANE by its name. This method should return none or exactly one INSANE, otherwise something is wrong.

- Call:
  - `insane`=*NAME*
- HTTP Replies (header | body):
  - 200 OK | XML representation of the INSANE information
  - 400 Bad Request | HTML resource
  - 404 Not Found |
- Expectations:
  - a 400 error suggests a missing variable; this problem must be solved by the requester

I UNACCOUNTED-FOR EVALUATION SETTINGS

As mentioned in the proceedings, four settings were not considered for the evaluation of the performance and scalability aspects of the newly implemented components. However, these four settings may help to improve the quality of the network in the affected parts of the faculty building, so they are provided here. Ex ante, the results are not very pleasing as they show that about 50% of all packets are lost for 100 parallel competing requests. For 1000 and more competing requests over 70% of the packets are lost.

---

1. **LAN Setting**
   - The machines simulating the clients, a machine acting as an INSANE and the actual building server machine ('CARLOS') were connected in a local area network[161] within the faculty building of the Faculty of Computer Science at TUD.
   - *This setting is designed to run under laboratory conditions.*
   - The conceptual setup is displayed in Figure I.0.1.
2. **'CARLOS' setting**
   - The machines simulating the clients were placed out of campus in an apartment[162] as well as a server farm north-west of Dresden, Saxony. 'Carlos' acted as INSANE while also facilitating as building server.
   - *This setting is designed to put double load on 'CARLOS'.*
   - The conceptual setup is displayed in Figure I.0.2.
3. **the-tester.de setting**
   - The machines simulating the clients were placed out of campus in an apartment[162] as well as a server farm north-west of Dresden, Saxony. An actual INSANE at `http://insane.the-tester.de` was used. 'CARLOS' facilitated as building server.
   - *This setting is designed to simulate actual access behaviour.*
   - The conceptual setup is displayed in Figure I.0.3.
4. **hara.tc setting**
   - The machines simulating the clients were placed out of campus in an apartment[162] as well as a server farm north-west of Dresden, Saxony. An actual INSANE at `http://insane.hara.tc` was used. 'CARLOS' facilitated as building server.
   - *This setting is designed to simulate actual access behaviour.*
   - The conceptual setup is displayed in Figure I.0.4.

[161] The virtual clients running on the VPS in a server farm were connected from outside the LAN.
[162] Connected to the internet via a DSL-6000 connection.

All four settings rely on 'CARLOS' as building server. This is due to the fact that only four buildings are currently supported by the MapBiquitous project, and all of them are hosted on 'CARLOS'. Further, time and financial resources did not permit to set up another building server, even when simply copying the data from 'CARLOS'.

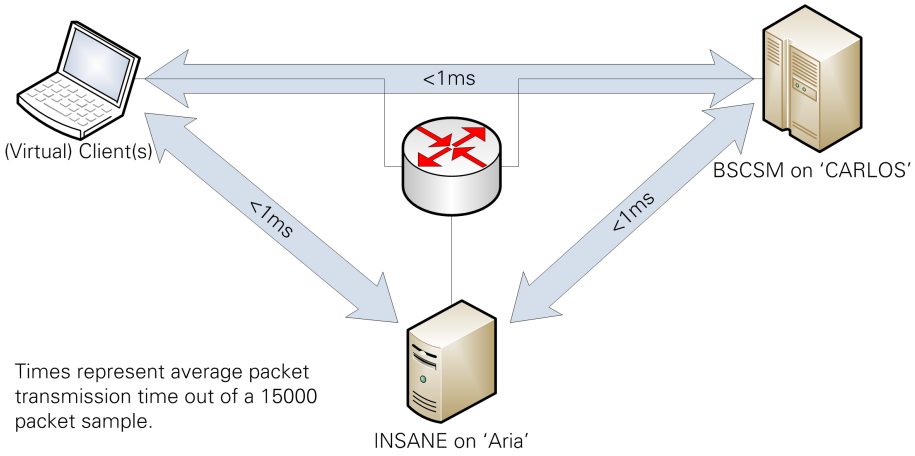


Figure I.0.1: Conceptual layout of the LAN setting

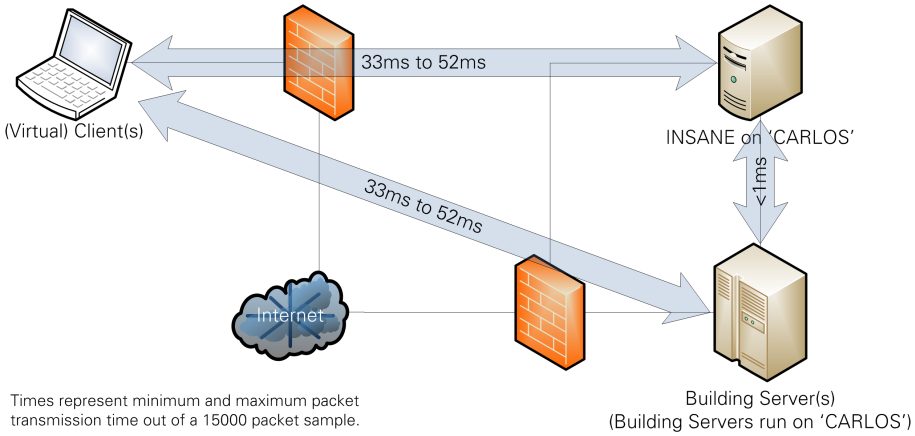


Figure I.0.2: Conceptual layout of the 'CARLOS' setting

The basic idea starting into the evaluation of resource use and communication load is to prove that the results are strongly correlated to the hardware and general load of the servers. This would prove comparability of the scalability results to general web servers, and as such would allow to simply apply findings of web server tests to the MapBiquitous server components. In order to generate comparable results for getter as well as setter access, two types of series of measurements were conducted for all four settings described above.

1. A series-type focussed on a WFS request, namely '`?SERVICE=wfs&VERSION=1.0.0 &REQUEST=GetFeature&TYPENAME=tud_inf:TUD_INF_G&`' addressed at the building server 'tud_inf:TUD_INF' was evaluated.
2. A series-type focussed on the creation of a WLAN fingerprint, namely consisting of six samples of length 177 Byte each, also addressed at 'tud_inf:TUD_INF' was evaluated.

The interfaces descriptions as given in Appendix H apply for both types. System loads were directly recorded from '`uptime`' on the Linux-machines and '`typeperf "\Processor(_Total)\% Processor Time"`' on the Windows-machines, and the network statistics were retrieved via '`ab`'[163] and 'Wireshark'.[164]

[163] `http://httpd.apache.org/docs/2.2/programs/ab.html` – Accessed 12 October 2012

[164] Wireshark 1.8.3 was used in recording mode. The log-files were evaluated after the measurements, as live-evaluation would have falsified the results by adding further load to the machines.

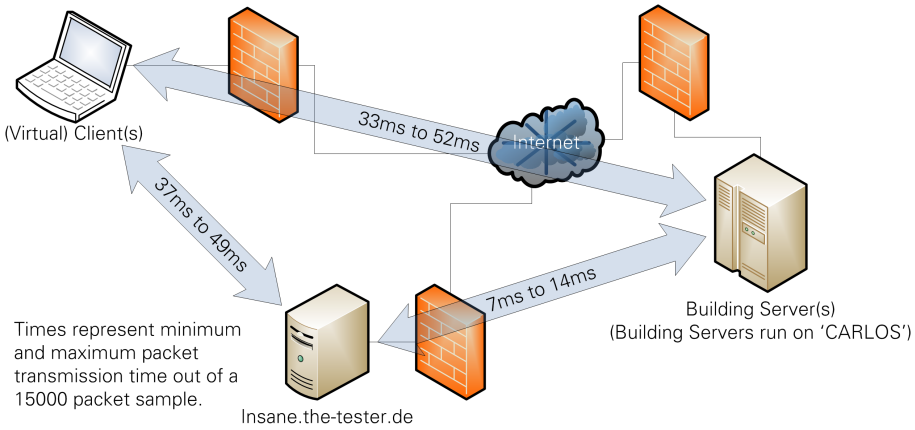


Figure I.0.3: Conceptual layout of the the-tester.de setting

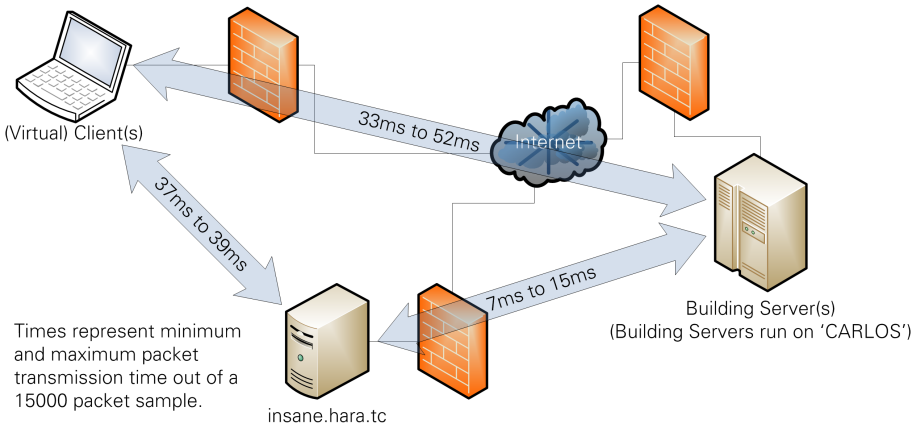


Figure I.0.4: Conceptual layout of the hara.tc setting

Both series types were conducted for different amounts of parallel client requests and were repeated ten times. The results were then averaged in one representative result table for each setting and type.

I.0.2 WFS Request Series

The results for the first series type can be found in tables F.5.1 to F.5.4. Additionally, they are visualised in figures I.0.5 to I.0.8.

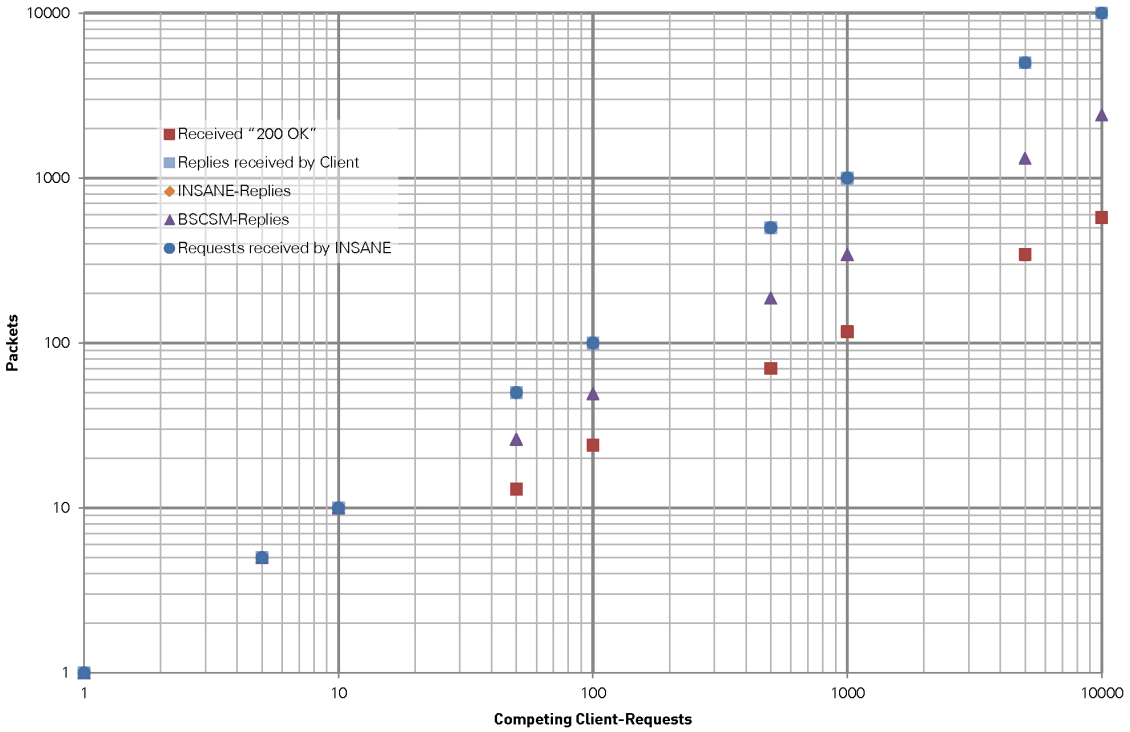


Figure I.0.5: Packet transmission statistics for WFS requests and replies in the LAN setting

The results of the first series-type clearly show an exponential correlation between the amount of parallel client requests and the packet losses as well as the successful replies. This is not surprising as the communication follows a linear path, from the client to the INSANE, to the BSCSM, to the WFS-service, back to the BSCSM, back to the INSANE and finally back to the client. Comparing all four settings, the major source of packet loss seems to be the interface to 'CARLOS', where the BSCSM seems to miss out on packets as well as the WFS-service which also seems to miss out on packets. Analysing even deeper into the numbers, two critical thresholds can be identified; one at approximately 40 parallel requests and one at approximately 70 parallel requests.

Of the two thresholds, the first seems to be the maximum amount of requests 'CARLOS' can handle in parallel. Surpassing the threshold, 'CARLOS' starts dropping packets from the network queue. Strangely, the same threshold applies for internal (i.e. localhost/127.0.0.1) communication within 'CARLOS'. – The surmise is, that the high base idling load of 'CARLOS' is responsible. However, the internals of the processing queue, etc. of 'CARLOS' were impossible to disclose.

Then, the second threshold is definitely the maximum amount of requests the VPS servers running the-tester.de and hara.tc can handle without packet loss. This coincides totally with the

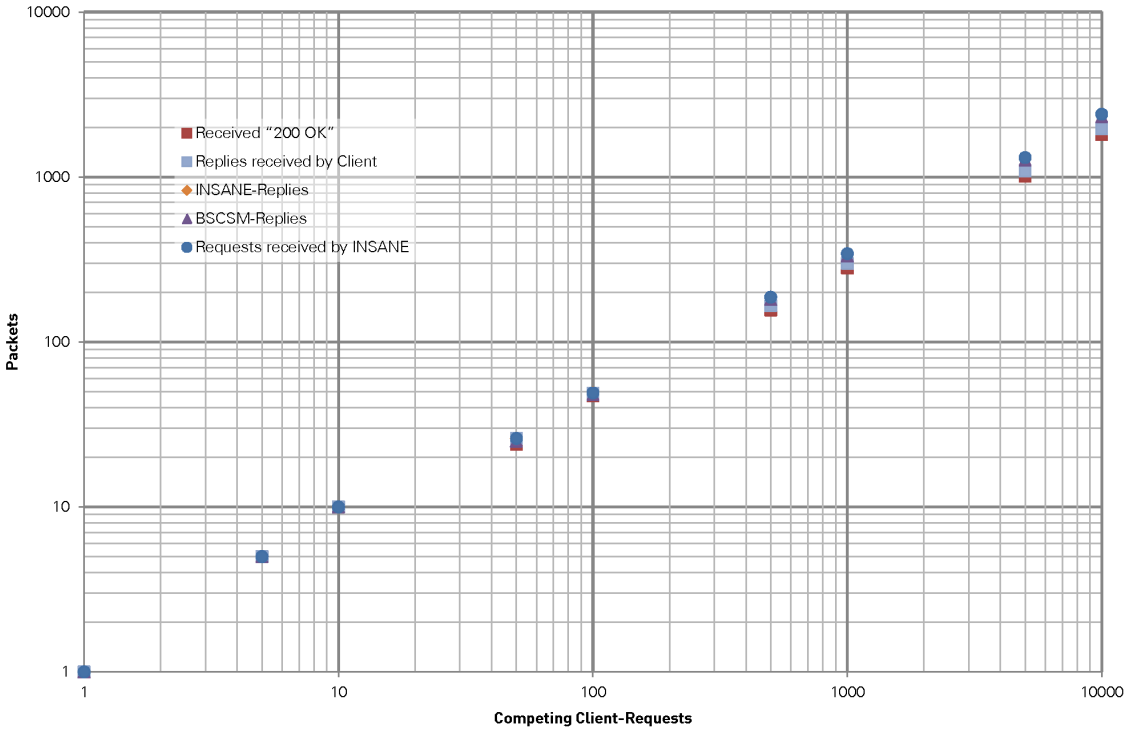


Figure I.0.6: Packet transmission statistics for WFS requests and replies in the 'CARLOS' setting

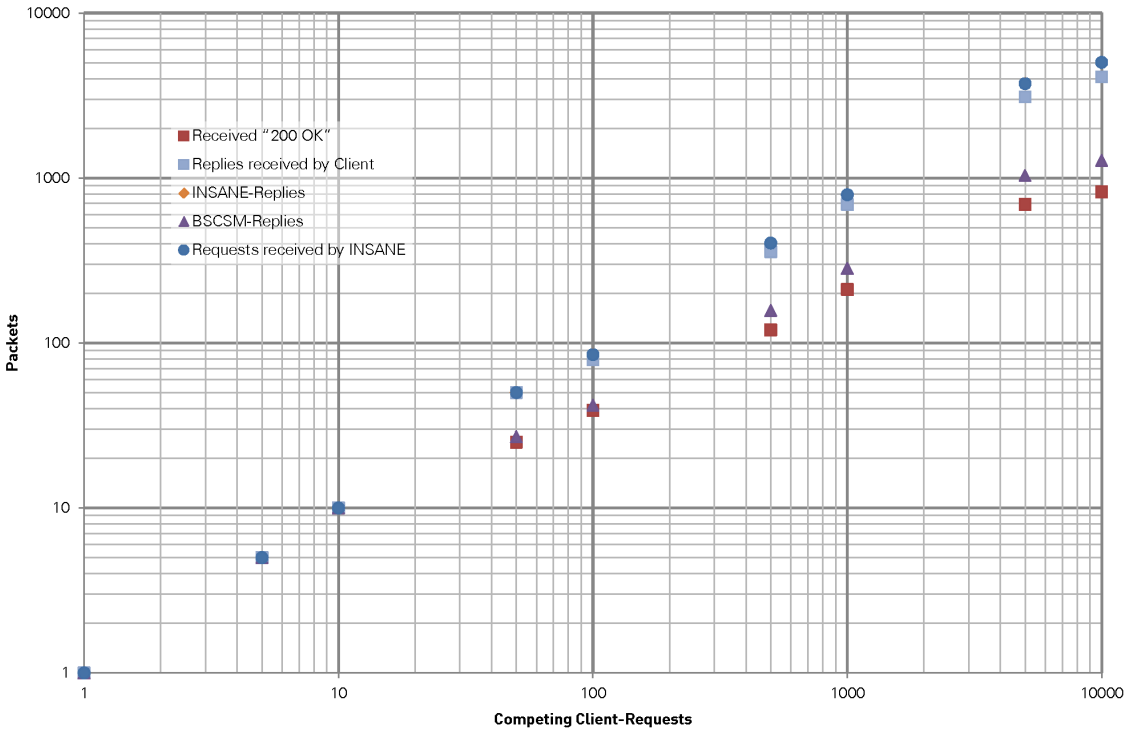


Figure I.0.7: Packet transmission statistics for WFS requests and replies in the the-tester.de setting

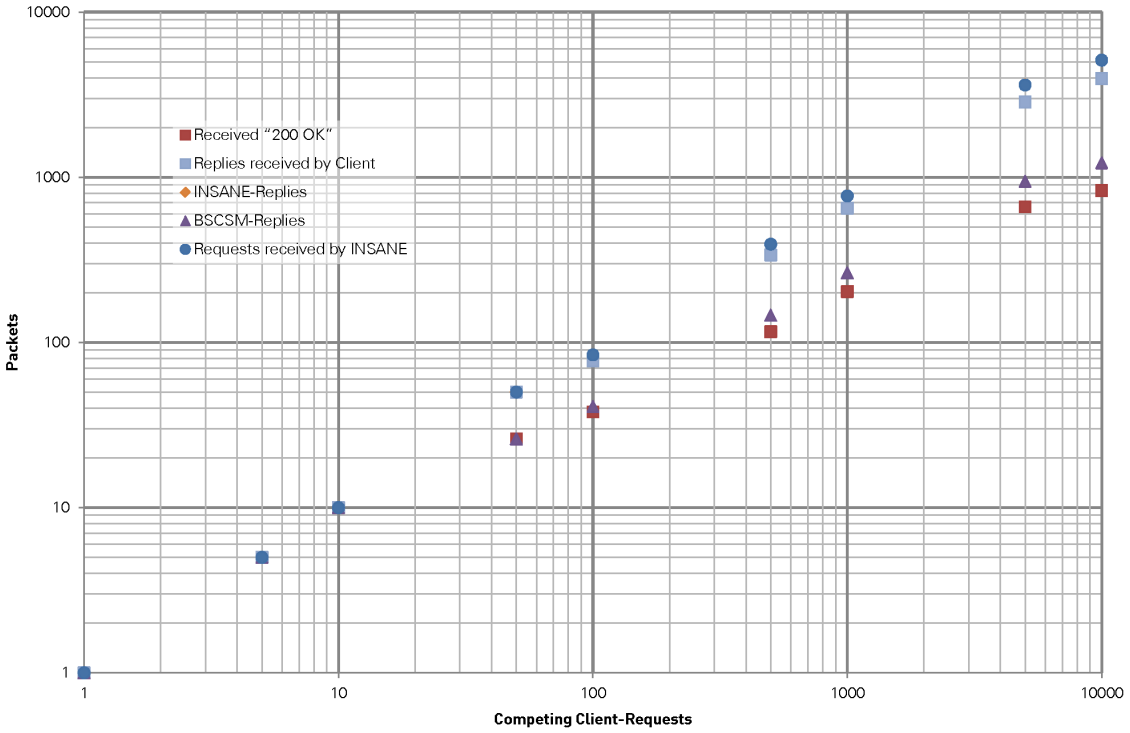


Figure I.0.8: Packet transmission statistics for WFS requests and replies in the hara.tc setting

normal operational specifics of the VPS. However, even though packets are dropped from the network queue, a maximum of about 5000 requests and replies can be handled by the deployed INSANEs, which coincides with diverse recommendations for similar server configurations. As there are hundreds of such recommendations that this can really be considered general knowledge, the interested reader shall exemplary be referred to the manual for the Apache httpd web server at `http://httpd.apache.org/docs/2.0/`.

Summarising, the INSANE as implemented follows the operating figures of any general web server. This is not surprising as it is running on PHP in Apache httpd. Even without having the implemented BSCSM deployed on servers other than 'CARLOS', it is fair to apply the same result to the BSCSM as it shares the same code base with the INSANE. The actual bottleneck seems to be 'CARLOS' and possibly the WFS server which runs on Java in Apache Tomcat. Therefore, additional evaluation should be conducted with different hardware, i.e. deploy WFS server and BSCSM on a machine other than 'CARLOS'. However, this can only be considered a concluding recommendation, as time and financial resources do not permit such course of action in the moment.

I.0.3 Fingerprinting Request Series

Applying the same operating figures to the second series as in section 8.2.2, the average size of a fingerprinting packet sent from the client to the INSANE is 1061 Byte. The forwards from the INSANE to the building server has an average size of 1172 Byte and within the building server 8 requests of a total of 1083 Byte are sent. Therefore, the evaluation of the fingerprinting series was conducted using actual fingerprinting samples matching the average size. Especially the fragmentation of the original fingerprinting request sent by the client into eight requests by the BSCSM is worth investigating. Presumably, the fragmentation leads to information loss, as the fingerprinting server requires one packet to start the fingerprinting, six packets for the samples and one final packet closing the fingerprinting.

The results for the second series-type can be found in in tables F.5.5 to F.5.8; they are visualised in figures I.0.9 to I.0.12.

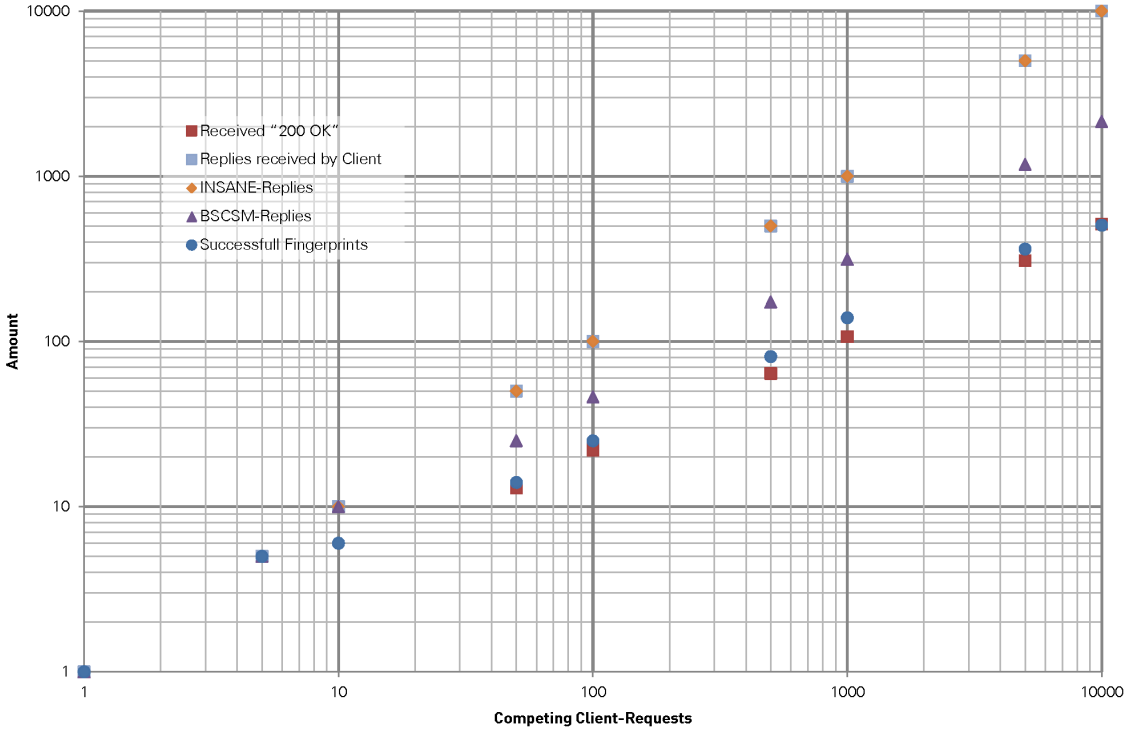


Figure I.0.9: Packet transmission statistics for fingerprinting requests and replies in the LAN setting

As true for the WFS-series, these results clearly show an exponential correlation between the amount of parallel client requests and the packet losses as well as the successful replies. Once again, this is not surprising as the communication follows the same linear path as the WFS communication. However, a discrepancy between '`HTTP/1.1 200 OK`' replies and actually successfully created fingerprints is obvious. Especially in the LAN setting not all created fingerprints have a corresponding success-reply sent[165] to the clients.

With a certain degree of likelihood, the discrepancy can only be explained by the interplay of a communication error and an implementation error. The most likely explanation would be that not all samples were received by the fingerprinting server in the course of the evaluation. Then, the fingerprinting server would have initialised fingerprint-creation after receiving the initial packet and would have successfully closed the fingerprint after receiving the final packet. In between, from the six samples enclosed by the initial and final packet, not all would have been received. This would be the communication error. However, the initial packet of the fingerprinting procedure clearly transmits the amount of samples to be expected to the fingerprinting server. Therefore, it seems to be a reasonable assumption to blame an implementation error in [Gru12], as the amount of samples seem to be ignored.

Another explanation, however very unlikely[166], could be the loss of packets – especially initial and final fingerprinting packets – combined with the very unlikely choosing of the same random integer by different clients at the same time.

[165] 'Sent' is the correct choice of phrasing here, as packets lost while transmission and hence even fewer being received by the clients are not even considered, yet.
[166] Assuming evenly distributed selection of random integers, the likelihood of this explanation is 1 : 2147483647.5.

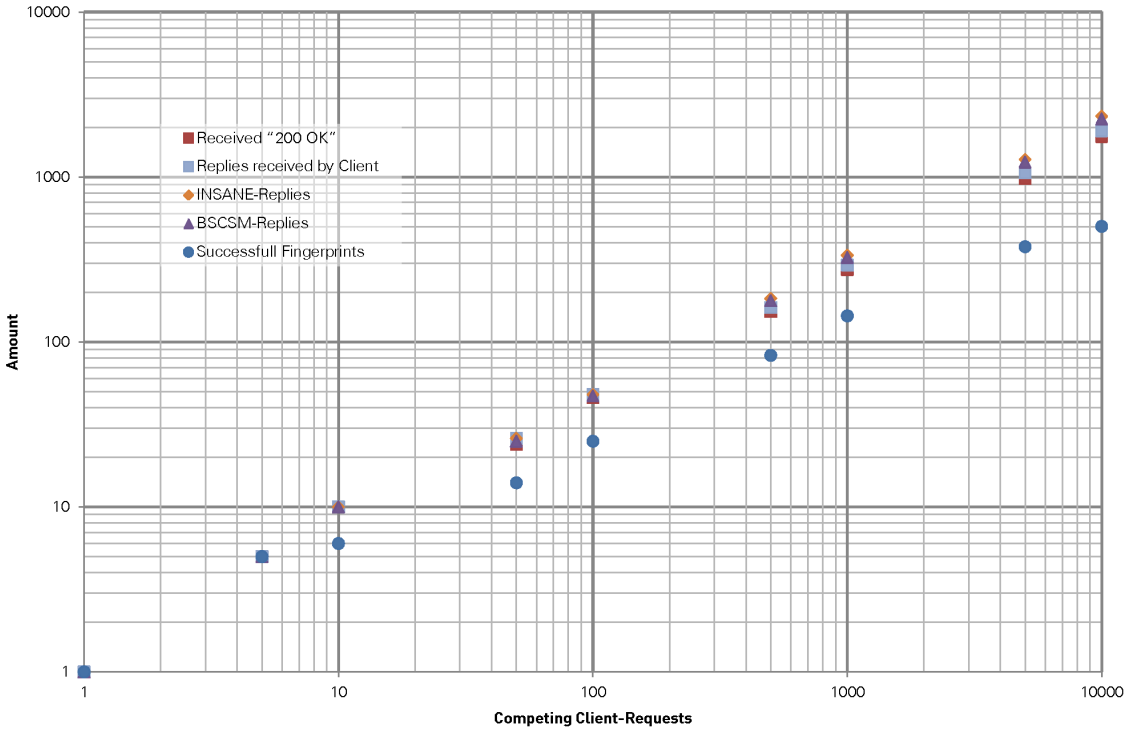


Figure I.0.10: Packet transmission statistics for fingerprinting requests and replies in the 'CARLOS' setting

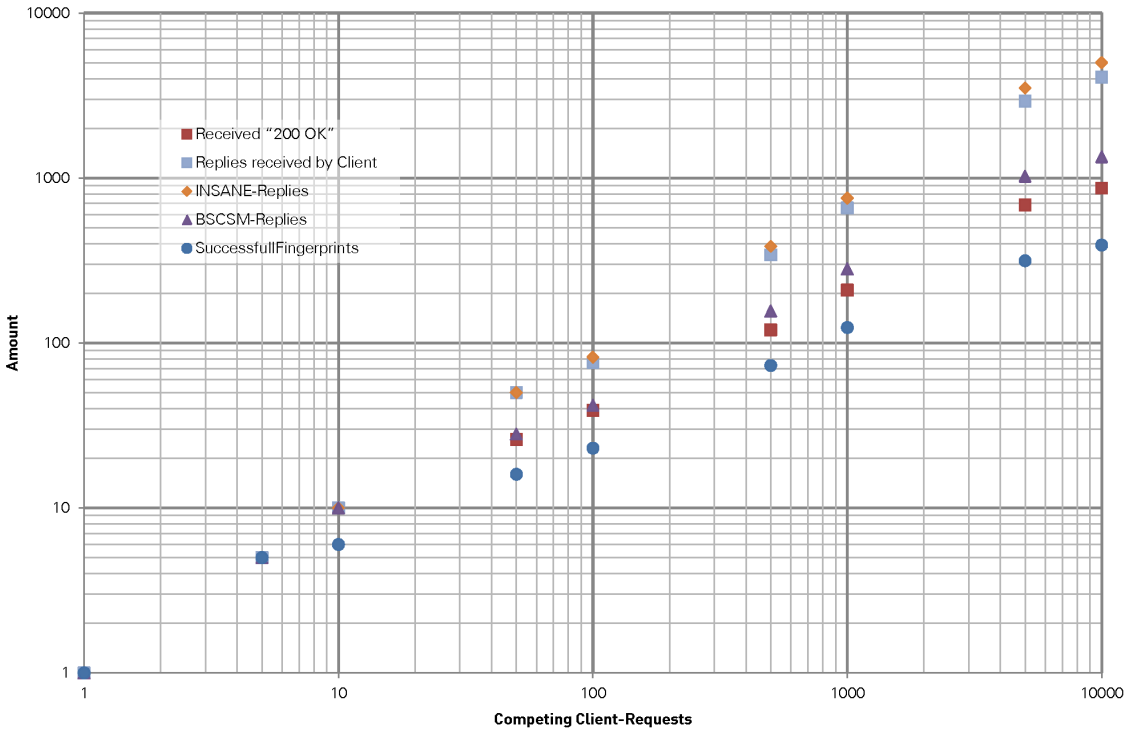


Figure I.0.11: Packet transmission statistics for fingerprinting requests and replies in the the-tester.de setting

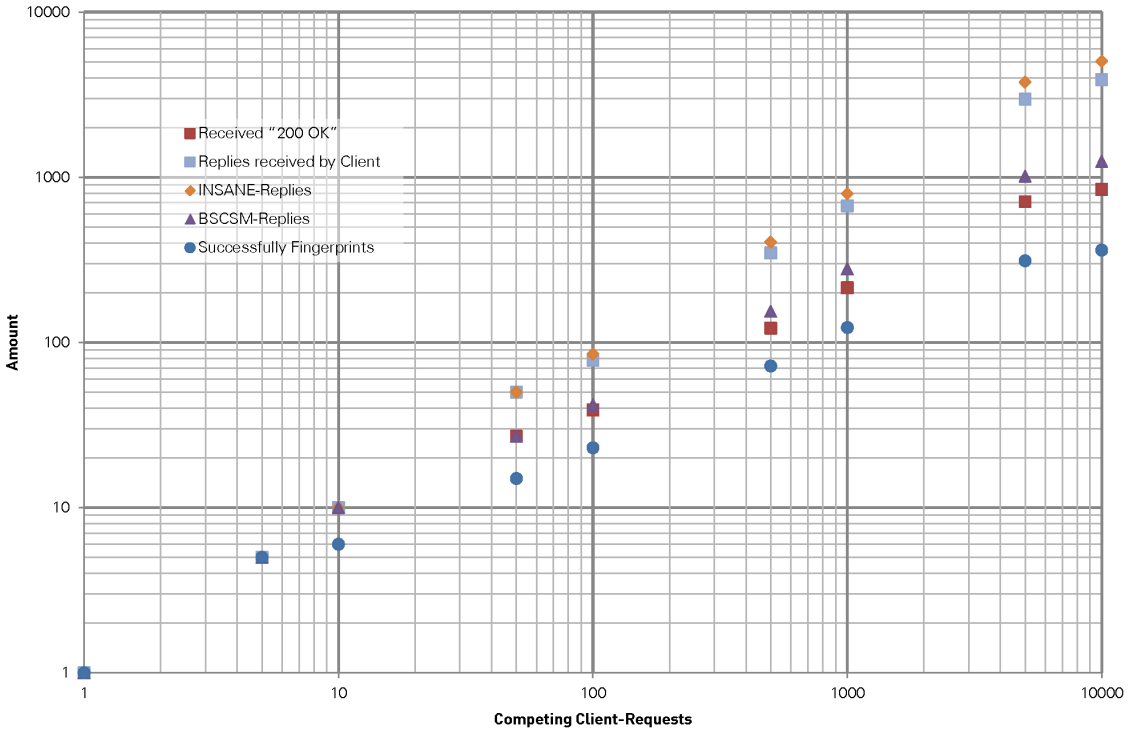


Figure I.0.12: Packet transmission statistics for fingerprinting requests and replies in the hara.tc setting

Comparing all four settings, the major source of packet loss once again seems to be the interface to 'CARLOS', where the BSCSM seems to miss out on packets as well as the fingerprinting server which also seems to miss out on packets. Very similar thresholds as found for the WFS-series can be identified. Therefore, the same conclusions apply.

Summarising, the INSANE and the BSCSM as implemented both follow the operating figures of any general web server. The actual bottleneck remains to be 'CARLOS' or the network it resides in, and possibly a deficient implementation of the fingerprinting server. – Additional evaluation should be conducted to find the source of the abnormal packet loss rate.