Project no. 265432

EveryAware

Enhance Environmental Awareness through Social Information Technologies

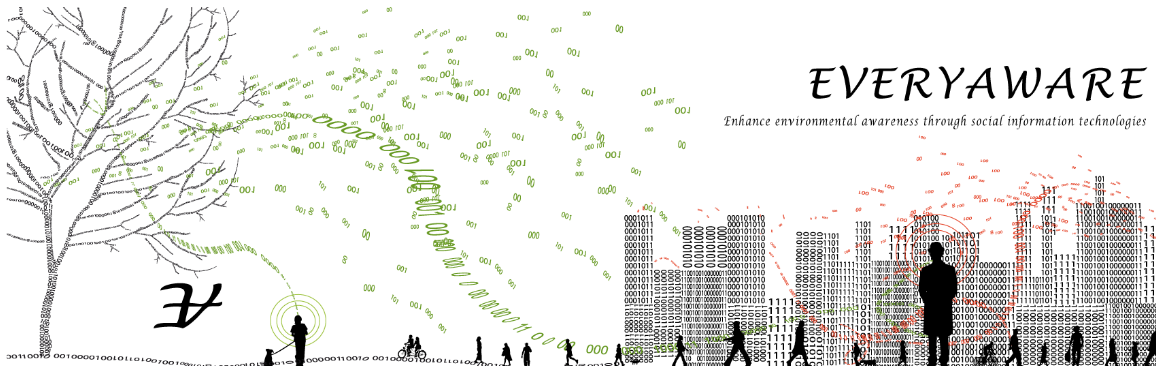
http://www.everyaware.eu

Seventh Framework Programme (FP7)

Future and Emerging Technologies of the Information Communication Technologies (ICT FET Open)

D2.1: First prototype of and interim report on web-based infrastructure

---



Period covered: from 01/03/2011 to 31/08/2012
Start date of project: March 1st, 2011
Due date of deliverable: Aug 31st, 2012
Distribution: Public

Date of preparation: 31/08/2012
Duration: 36 months
Actual submission date: Aug 31st, 2012
Status: Final

Project coordinator: Vittorio Loreto
Project coordinator organisation name: Fondazione ISI, Turin, Italy (ISI)
Lead contractor for this deliverable: Gottfried Wilhelm Leibniz Universität Hannover (LUH)

2012 © Copyright lies with the respective authors and their institutions

Executive Summary

A characteristic of social information technologies as they are used within the EveryAware project is that they often involve very large amounts of data. In fact, the collection, storage and analysis of different kinds of data within these systems is a crucial point and also an asset, e.g., for companies like Facebook[1]. As a consequence, in order to pave the way towards behavioral shifts within large citizen populations, methods and techniques of acquiring and handling data play a central role. The design of web-based infrastructures for this purpose has a great influence both on data quantity and quality, and hence also on the additional value which can be generated by analyzing the resulting datasets.

The data in the context of the EveryAware project can be divided into two classes, namely (i) *objective data*, which stems mainly from sensors and captures things like sound intensity or gas concentration, and (ii) *subjective data* which comprises reactions of humans faced with particular environmental conditions. This deliverable is structured along these two dimensions.

Ad (i): UBICON is a framework which is used to build a high-performance data storage infrastructure. The resulting system can be seen as the primary data backend for most of the applications developed within the EveryAware project. It comprises among others different endpoints e. g., for noise and air measurements, well-defined interfaces for exchanging data with web applications and data processors for extracting particular information from the raw inputs.

Ad (ii): Gaming is a well-established way to engage humans in experiments. Although the idea of *crowdsourcing* is already implemented in systems like Amazons Mechanical Turk, the design of social games requires more complex interfaces which are hardly realizable in current implementations. For this purpose, we introduce *Experimental Tribe* or XTribe, a platform for web-based experiments and social computation. Its goal is to allow researchers to realize their own experiments with minimal efforts, leading towards the Web as a "laboratory" to perform studies.

In summary, the data storage system and the gaming platform are the two main components of the EveryAware web-based infrastructure, which complement each other by addressing specific goals in the context of collecting, storing and analyzing relevant environmental data.

Outline of the document

In Chapter 2, we describe the UBICON framework which is used to build a high-performance data storage infrastructure. The following Chapter 3 introduces *Experimental Tribe* or XTribe, a platform for web-based experiments and social computation.

Dissemination of the Results

The EveryAware frontend based on the Ubicon framework is online at `http://http://cs.everyaware.eu`, and XTribe can be reached at `http://www.xtribe.eu/`. The social game ”Joe's City race" has been presented at the European Conference on Complex Systems (ECCS12), September 2012,

[1] `http://www.facebook.com`

within the workshop “Complexity paradigms for Smart, Green and Integrated Transport". The same game has participated in the Idea Lab workshop, in June 2012, where it has been presented and displayed in the Science Gallery in Dublin, Ireland, within the “Hack the City” exhibition.

2012 © Copyright lies with the respective authors and their institutions

Contents

1 Overview **8**

2 Data Storage Infrastructure **10**
2.1 Introduction 10
2.2 Database 11
2.2.1 Table dataairqualityapp 11
2.2.2 Table datawidenoise 12
2.2.3 Table events 14
2.2.4 Table measurements 15
2.2.5 Table pendinguser 15
2.2.6 Table tags 16
2.2.7 Table twitter_status 17
2.2.8 Table user 17
2.3 REST Server 19
2.3.1 Responses and Errors 19
2.3.2 Data Collection 19
2.3.3 Noise Endpoints 20
2.3.4 Air Endpoints 21
2.3.5 Webapp Endpoints 22
2.4 Data Processor 28
2.4.1 Request Parser 29
2.4.2 Location Appender 30
2.4.3 OpenStreetMaps Appender 31
2.4.4 Tag Extractor 31
2.5 Ubicon 31
2.5.1 Overview 31
2.5.2 WideNoise 32

3 Gaming Platform **45**
3.1 Introduction 45
3.2 Experimental Tribe 47
3.2.1 Blindate 47
3.2.2 Joe’s City Race 50
3.3 Technical details 55
3.3.1 Basic entities 56
3.3.2 Technologies for development and communication 56

3.3.3 Game setup 57
3.4 Further developments of the platform 58

2012 © Copyright lies with the respective authors and their institutions

List of Figures

2.1 dataairqualityapp table schema 11
2.2 datawidenoise table schema 13
2.3 events table schema 14
2.4 measurements table schema 15
2.5 pendinguser table schema 16
2.6 tags table schema 16
2.7 twitter_status table schema 17
2.8 user table schema 18
2.9 Component Overview of the Data Processor 29
2.10 Static pages of WideNoise event of the “noise” category. 33
2.11 Social sharing functionality on the “WideNoise” event’s front page. 34
2.12 Front page statistics for the WideNoise event of the “noise” category. 35
2.13 User statistics for the WideNoise event of the “noise” category. 36
2.14 Coverage page of the “WideNoise” event. 38
2.15 Map page of the “WideNoise” event. 39
2.16 Data representations on the map page of the “WideNoise” event. 40
2.17 Elements on the map page of the “WideNoise” event. 41
2.18 Tagcloud functionality on the map page of the “WideNoise” event.. 42
2.19 Profile page. 44

3.1 A screenshot of the XTribe homepage. 46
3.2 A set of screenshots of the Blindate interface. 48
3.3 Rome map of focal points drew with the position guessed in Blindate. Redder points indicate an higher density of guess, corresponding to a focal point. 49
3.4 In the left graph, the ratio of winning matches at each turn (55% of match at the first turn; of the remaining 45% the 35% matched at the second turn and so on). In the right graph, the average distance between the guess of the two players at each turn of the game. 49
3.5 Joe’s City Race: select location. 51
3.6 Joe’s City Race: single-player, phase I, no traffic information. 51
3.7 Joe’s City Race: single-player, phase II, traffic displayed as colours on the streets. . 52
3.8 Joe’s City Race: score for single-player game. 52
3.9 Joe’s City Race: multi-player. 53
3.10 Joe’s City Race: score for multi-player game. 54
3.11 Effect of the amount of traffic information on player performance. 54
3.12 Imitation during multi-player games. 55

3.13 Score for multi-player games. 56
3.14 Communication between the ET Server and the GM. 57

2012 © Copyright lies with the respective authors and their institutions

Chapter 1

Overview

A characteristic of social information technologies as they are used within the EveryAware project is that they often involve very large amounts of data. In fact, the collection, storage and analysis of different kinds of data within these systems is a crucial point and also an asset, e.g., for companies like Facebook[1]. As a consequence, in order to pave the way towards behavioral shifts within large citizen populations, methods and techniques of acquiring and handling data play a central role. The design of web-based infrastructures for this purpose has a great influence both on data quantity and quality, and hence also on the additional value which can be generated by analyzing the resulting datasets. Typical goals during the design process are:

- *Performance:* Because the involvement of large numbers of humans requires responsive interfaces and efficient server backends, all infrastructures must be carefully tuned for high-performance requirements of processing large amounts of data in a parallel fashion.
- *Management:* The setup and technical realization of experiments and studies among citizens often implies strong efforts on the side of scientists and experimenters. As a consequence, it is desirable to provide reusable and configurable experimentation platforms which can easily be managed.
- *Correctness:* A large-scale collection of data can hardly be expected to provide only correct and consistent results. However, the reduction of noise from the very beginning (i.e., the concrete measurements) is desirable in order to provide a better basis for later analysis.

Broadly speaking, the relevant data in the context of the EveryAware project can be divided into two classes, namely (i) *objective data*, which stems mainly from sensors and captures things like sound intensity or gas concentration, and (ii) *subjective data* which comprises reactions of humans faced with particular environmental conditions. This deliverable is roughly structured along these two dimensions.

In Chapter 2, we start by describing how the UBICON framework is used to build a high-performance data storage infrastructure. The resulting system can be seen as the primary data backend for most of the applications developed within the EveryAware project. It comprises among others different endpoints e. g., for noise and air measurements, well-defined interfaces for exchanging data with web applications and (iii) data processors for extracting particular information from the raw inputs.

The UBICON framework itself is used as a modular and flexible framework for embedding customized components, based on standard software and open standards. These include e. g., authentication and authorization modules as well as the aforementioned data processors. Technically,

[1] http://www.facebook.com

all data is stored within a MySQL database. To be clear, the storage infrastructure does not contain exclusively objective data (from sensors), but also subjective data gathered e. g., from users tagging measurements.

The following Chapter 3 shifts perspective and focusses solely on *subjective data*, more precisely on *gaming* as a well-established way to engage humans in experiments. Although the idea of *crowdsourcing* is already implemented in systems like Amazons Mechanical Turk, the design of social games requires more complex interfaces which are hardly realizable in current implementations. For this purpose, we introduce *Experimental Tribe* or XTribe, a platform for web-based experiments and social computation. Its goal is to allow researchers to realize their own experiments with minimal efforts, leading towards the Web as a "laboratory" to perform studies.

The core idea behind XTribe is to offer a set of readily useable standard components, which are used within a great bandwidth of different experiments. Those include e. g., user handling, interface hosting or security issues. Similar to the data storage architecture, it has a modular structure, which allows the researcher to focus on his core questions by hiding away most of the complexity associated with running a web-based experiment. On the other hand, it is furthermore intended to serve as a "basin of attraction" for people willing to participate in experiments.

In summary, the data storage system and the gaming platform are the two main components of the EveryAware web-based infrastructure, which complement each other by addressing specific goals in the context of collecting, storing and analyzing relevant environmental data.

2012 © Copyright lies with the respective authors and their institutions

Chapter 2

Data Storage Infrastructure

2.1 Introduction

The web-based infrastructure of the EveryAware platform is implemented using the UBICON software framework[1] for implementing social and ubiquitous applications. UBICON has been developed jointly in the EveryAware project and in the VENUS project[2] where it is used for additional social application: in the CONFERATOR[3] system - for guiding and supporting participants during a conference, and for the MYGROUP[4] system for computer-assisted ubiquitous social networking in working groups.

UBICON provides a modular and flexible framework for embedding customized components; it is based on standard software components, and applies open standards for the extension and access to the system.

From a technical perspective, the platform consists of an authentication and authorization component, a user management component, a (customizable) set of data processors that process the incoming data, a social extension component, and a storage architecture based on a MySQL database.[5] The set of data processors include, e. g., the localization component for determining the location of tags. For management and data access, UBICON features a flexible REST-based architecture. The components are tied together using the Spring framework[6] and can be deployed using a standard servlet container, e. g., Apache Tomcat.[7]

UBICON enables the organization of applications in several *events*. For example, for the WideNoise application, the data could be organized into different event schemes, as to support different experiments, if necessary. In the Conferator and MyGroup systems, for example, this functionality is used for providing different events corresponding to conferences (e.g., LWA 2010[8]/2011[9]/2012[10] or Hypertext 2011[11]), and for events focusing on different working group environments.

In the following sections, we first describe the technical implementation concerning the database backend, before we describe the web-based front-end in more detail.

[1] http://www.ubicon.eu/
[2] http://www.uni-kassel.de/eecs/iteg/venus/
[3] http://conferator.org/
[4] http://ubicon.eu/about/mygroup
[5] http://www.mysql.com/
[6] http://www.springsource.org/
[7] http://tomcat.apache.org/
[8] http://lwa2011.dke-research.de/
[9] http://lwa2011.dke-research.de/
[10] http://lwa2012.cs.tu-dortmund.de/
[11] http://www.ht2011.org/

2.2 Database

We use a MySQL 5.1.62 database that runs inside a VirtualBox. This section describes the table structure of our EveryAware database. Our database is not normalized and we do not use foreign keys in order to keep the performance of our server a high as possible.

2.2.1 Table dataairqualityapp

The airqualityapp Table contains all parsed reports received from the Air Quality App Android application. Most of the information is extracted directly from the report requests, some are added by our data processor (i.e., city, state, country, map_data).

A description about the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.1.

Figure 2.1: dataairqualityapp table schema

- id: A unique identifier to address a certain data set.
- measurement_id: The identifier that addresses the raw data in the measurements table.
- client: The name of the user agent which we extracted from the User-Agent HTTP header field.
- event: The name of the event in which's context the measurement was collected (e.g., “AirQualityApp”).
- ip: The source IP address from which we received the measurement.

2012 © Copyright lies with the respective authors and their institutions

- rawdata: The content of the received message as it was received (e.g., the whole JavaScript Object Notation (JSON)[12] object as String representation).
- request_ts: The time stamp when we received the corresponding request.
- uid: A string that identifies a device. This string is a 31 to 32 signs HEX MD5 hash string from iOS devices and a 16 digits long number from Android devices.
- session_id: Identifies a coherent stream of measurement data.
- device: Device identifier, behaves like a user agent for browsers.
- lat: The geographic latitude coordinate of the noise sample.
- lon: The geographic longitude coordinate of the noise sample.
- sample_ts: The UNIX timestamp of the sample.
- co_sensor[i]: Recorded value from the four CO sensors.
- no2_sensor[i]: Recorded value from the two NO_2 sensors.
- voc_sensor1: Recorded value from the VOC sensor.
- o3_sensor1: Recorded value from the O_3 sensor.
- temperature: Recorded value from the temperature sensor.
- humidity: Recorded value from the humidity sensor.
- user_data_[i]: Subjective data expressed by the user.
- city: The name of the city where the measurement was taken.
- state: The name of the state where the measurement was taken.
- country: The name of the country where the measurement was taken.
- map_data: A pre-processed JSON object containing all information required from WideNoise to display the measurements on its map if user_data_1 information are present.

2.2.2 Table datawidenoise

The datawidenoise Table contains all parsed reports received from the WideNoise iOS and Android applications. Most of the information are extracted directly from the report requests, some are added by our data processor (i.e., city, state, country, map_data).

A description of the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.2.

- id: A unique identifier to address a certain data set.
- event: The name of the event in which's context the measurement was collected (e.g., "Wide-Noise").
- client: The name of the user agent which we extracted from the User-Agent HTTP header field.

[12] http://json.org/

datawidenoise
id BIGINT(20)
client TEXT
event VARCHAR(32)
ip VARCHAR(32)
request_ts DATETIME
uid CHAR(32)
device VARCHAR(64)
lat DOUBLE
lon DOUBLE
sample_ts DATETIME
duration DOUBLE
average_db DOUBLE
average_raw DOUBLE
samples TEXT
tags TEXT
perception_feeling DOUBLE
perception_disturbance DOUBLE
perception_isolation DOUBLE
perception_artificiality DOUBLE
location_precision VARCHAR(4)
location_accuracy DOUBLE
user_estimate DOUBLE
city VARCHAR(128)
state VARCHAR(128)
country VARCHAR(128)
map_data TEXT
Indexes
PRIMARY

Figure 2.2: datawidenoise table schema

- ip: The source IP address from which we received the measurement.
- rawdata: The content of the received message as it was received (e.g., the whole JSON object as String representation).
- request_ts: The time stamp when we received the corresponding request.
- uid: A string that identifies a device. This string is a 31 to 32 signs HEX MD5 hash string from iOS devices and a 16 digits long number from Android devices.
- device: Device identifier, behaves like a user agent for browsers.
- lat: The geographic latitude coordinate of the noise sample.
- lon: The geographic longitude coordinate of the noise sample.
- sample_ts: The UNIX timestamp of the sample.
- duration: The duration of the sample, in seconds. This could be 5, 10, or 15 seconds, depending on how often the user extended the measurements.
- average_db: The average noise value, expressed in dB, visualized to the user on their display.
- average_raw: The average noise value, expressed as the internal microphone API raw data, from which average_db is derived.
- samples: The list of all the samples taken over the sampling period of time (one every 0.5 seconds). It is expressed as raw data directly from the internal microphone API and stored as a serialized JSON array.
- tags: Contains all tags expressed by the user as a serialized JSON array.

2012 © Copyright lies with the respective authors and their institutions

- perception-feeling: The user's perception about the his emotion for the noise, from love (0.0) to hate (1.0).
- perception-disturbance: The user's perception about the disturbance of the noise, from calm (0.0) to hectic (1.0).
- perception-isolation: The user's perception about the social environment during the sampling, from alone (0.0) to social (1.0).
- perception-artificiality: The user's perception about the artificiality of the noise, from nature (0.0) to man-made (1.0).
- location_precision: Names the source of the location information (e.g., "gps").
- location_accuracy: Defines how accurate in meters the location information is.
- user_estimate: The guessed noise level from the user.
- city: The name of the city where the measurement was taken.
- state: The name of the state where the measurement was taken.
- country: The name of the country where the measurement was taken.
- map_data: A pre-processed JSON object containing all information required from WideNoise to display the measurements on its map.

2.2.3 Table events

The events table is used to control multiple event instance on our web application. Currently there is just the WideNoise event hosted on cs.everyaware.eu, but there will be additional ones as soon as the sensor box is finished. The stored information are used to define number, type, appearance, and basic functions for those web applications.

A description of the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.3.

Figure 2.3: events table schema

- id: The intern event specifier.
- category: The event category for this event (e.g., "noise" in the case of WideNoise).
- name: A human-readable name for this event (e.g., "EveryAware – WideNoise").
- description: A short description of this event.
- ordering: Defines in which order these events should appear in the menu.
- active: Enables or disables this event.
- afterlogin: Defines where the user will be redirected to after signing in.

2.2.4 Table measurements

The measurements table is the raw data storage. All data requests that are received from the REST server are stored to this table without any parsing processes in between. These information will be extracted by the data processor and parsed to their corresponding productive data tables (i.e., dataairqualityapp, datawidenoise, tag, twitter_status). The data are associated to the events using the event column. This table also holds information about the parsing state of the corresponding measurement that contains weather the data set was processed, processed successfully, and enriched successfully (if required).

A description of the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.4.

Figure 2.4: measurements table schema

- user: The user name of the owner of this measurement. Currently, this column is not used, since both EveryAware applications are not bound to a user name.
- client: The name of the user agent which we extracted from the User-Agent HTTP header field.
- event: The name of the event in which's context the measurement was collected (e.g., "Wide-Noise").
- format: Defines the format of the measurement which is stored in the rawdata column (e.g., "json").
- ip: The source IP address from which we received the measurement.
- header: The whole HTTP header of the received request.
- rawdata: The content of the received message as it was received (e.g., the whole serialized JSON object).
- ts: The time stamp when we received the corresponding request.
- id: A unique identifier to address a certain data set.
- state: A control flag that is used by the data processor the determine what to do with this report (see Section 2.4).

2.2.5 Table pendinguser

After a user finished the registration process, his data will be written into the pendinguser table until the registration is confirmed by an activation mail. We stored the data and time when the

2012 © Copyright lies with the respective authors and their institutions

registration attempt happened to be aware of the freshness of the corresponding process. The data will be removed after the account has been confirmed.

A description about the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.5.

Figure 2.5: pendinguser table schema

- email: The email address of the user.
- password: The chosen password as MD5 hash.
- widenoiseid: The device identifier of the WideNoise device if present.
- confirmkey: The generated key that was sent to the user to be compared with his answer.
- registered_at: The time and date when the user tried to register his / her account.

2.2.6 Table tags

The data processor extracts the tag information from the parsed WideNoise reports in the data-widenoise table. These are stored separately into the tags table where they are enriched with some additional information from the user table. Therefore the web application can access every single tag entity to display it on the map view.

A description about the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.6.

Figure 2.6: tags table schema

- measurement_id: The id of the corresponding report.
- lat: The latitude coordinate where this tag was attached.
- lon: The longitude coordinate where this tag was attached.
- name: The tag itself (could contain white spaces).
- uid: The device identifier of the WideNoise source device.

- username: The name of the user that transmitted this tag if he / she has an account in our system.
- event: The name if the related event (e.g., “WideNoise”).

2.2.7 Table twitter_status

There are entries in the measurements table that are neither WideNoise data nor Air Quality App data. Currently, these are Twitter status messages collected from the WideNoise application and are therefore associated to the WideNoise event. These data are collected if the WideNoise user connects his application with his Twitter account; hereby, status messages from this user’s timeline are collected.

A description about the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.7.

Figure 2.7: twitter_status table schema

- uid: The device identifier of the transmitting WideNoise device.
- timestamp: Time and date when this status message was posted on Twitter.
- user: The user name of the corresponding user in our system.
- text: The Twitter screen name.

2.2.8 Table user

The user table contains all personal information about the registered users of EveryAware. There are three system-relevant pieces of information about the corresponding user (i.e., name, display-name, and password). A user can use email or display name together with the password to perform the log-in. The name is just for system-internal references and cannot be changed by the user.

A description about the table structure and the meaning of all of its columns is given in the following list as well as in Figure 2.8.

- name: The internal user name.
- displayname: A name to be displayed on the web page.
- password: The user’s personal MD5-hashed password.
- title: The title of this user.
- forename: The user’s forename.
- surname: The user’s surname.
- birthday: The birthday of the user.

2012 © Copyright lies with the respective authors and their institutions

user
name VARCHAR(32)
displayname VARCHAR(6)
password VARCHAR(32)
title VARCHAR(32)
forename VARCHAR(64)
surname VARCHAR(64)
birthday DATE
gender CHAR(1)
picture VARCHAR(256)
spoken_languages TEXT
education TEXT
email VARCHAR(128)
tel VARCHAR(32)
fax VARCHAR(32)
street VARCHAR(64)
zip_code VARCHAR(8)
city VARCHAR(64)
country CHAR(2)
icq VARCHAR(128)
jabber VARCHAR(128)
msn VARCHAR(128)
skype VARCHAR(128)
bibsonomy VARCHAR(128)
facebook VARCHAR(128)
flickr VARCHAR(128)
googleplus VARCHAR(128)
linkedin VARCHAR(128)
researchgate VARCHAR(128)
twitter VARCHAR(128)
xing VARCHAR(128)
10 more...
Indexes
PRIMARY
unique_mail
unique_displayname

Figure 2.8: user table schema

- gender: The user's gender (i.e., "female", "male", or "other").
- picture: Contains a information if the user has an own picture or uses the default one.
- spoken_languages: A list of the user's spoken languages.
- education: The educational degree of this user.
- email: The user's email address.
- tel: The phone number of this user.
- fax: The fax number of this user.
- street: The street part of this user's address.
- zip_code: The ZIP code of the user's city.
- city: The city part of this user's address.
- country: The country part of this user's address.
- icq: This user's ICQ[13] number.
- jabber: This user's Jabber[14] address.
- msn: This user's MSN[15] number.

[13] http://www.icq.com/
[14] http://www.jabber.org/
[15] http://msn.com/

- skype: This user's Skype[16] name.
- bibsonomy: This user's BibSonomy[17] name.
- facebook: This user's Facebook[18] name.
- flickr: This user's flickr[19] name.
- googleplus: This user's Google+[20] number / name.
- linkedin: This user's LinkedIn[21] name.
- researchgate: This user's ResearchGate[22] name.
- twitter: This user's Twitter[23] screen name.
- xing: This user's Xing[24] name.

2.3 REST Server

We provide a RESTful interface to the EveryAware mobile applications as well as third party applications. The results are sent back to the caller as JSON objects. The API entrypoints are prefixed as `cs.everyaware.eu/api/`. Details are given in the sections below.

2.3.1 Responses and Errors

Our APIs returns status with a HTTP-like annotation. As a general guideline, here are the typical codes an API call may return. Further information on standard HTTP status codes can be found from the W3C:

- 200 OK: The request was successfully handled.
- 400 Bad Request: The server could not read the JSON object you provided.
- 500 Internal Server Error: The server encountered an unexpected condition which prevented it from fulfilling the request.

2.3.2 Data Collection

API endpoint: `[Event]/measurements` (POST)

The server accepts a POST request with a JSON object in its content for all noise and air data.

Hereby, measurements and tags are transmitted in two steps. First, the user client sends the measurements data while the user enters his tags. The tags are transmitted in a second request after the user confirms his input.

Both requests are handled in the same way. The REST server stores some general information, as they are, the HTTP header, source IP address, the user agent name, event name, content format,

[16] http://www.skype.com/
[17] http://www.bibsonomy.org/
[18] http://www.facebook.com/
[19] http://www.flickr.com/
[20] https://plus.google.com/
[21] http://www.linkedin.com/
[22] http://www.researchgate.net/
[23] http://twitter.com/
[24] http://www.xing.com/

2012 © Copyright lies with the respective authors and their institutions

the current time and date, as well as the JSON content of the request as a text. HTTP header, source IP address, and user agent name are retrieved from the HTTP request, format and event are retrieved from the request URI, and the JSON object is taken from the requests content. Those information are stored into the measurements table with state set to NULL and a unique ID.
The server will send back to the client a JSON response with a status code and the ID of the recorded noise / air sample. As an example:


```
{
  "status": "200",
  "id": "123456"
}
```


An id is not attached if the request was not successful.

2.3.3 Noise Endpoints

Requesting Map Data

API endpoint: `[Event]/noise/` (GET)

The server accepts a GET request with all the noise data. The parameter syntax is `lat=%f&lon=%f&lat_delta=%f&lon_delta=%f`

- lat and lon are the latitude and the longitude of the center of the map, respectively.
- lat_delta and lon_delta determine the visible map area (i.e. the zoom level). It's up to the server to decide if a region is too wide (or contains too many objects to be represented), in this case it would return a subset of all the objects or none of them.

The server responds with a JSON containing a status code and a data field containing an array of noise objects. For example:


```
{
  "average_db": 65.127933222222,
  "data": {
    "46047688470687": {
      "average_db": "56.846394",
      "geo_coord": [-122.406417, 37.785834],
      "timestamp": "1323033341",
      "duration": "10"
    },
    "86543046464888": {
      "average_db": "46.332741",
      "geo_coord": [-122.406417, 37.785834],
      "timestamp": "1321006236",
      "duration": "5"
    }
  }
}
```


Where:

- The id of each data set is used a JSON field name for the map object.
- geo_coord: The latitude and longitude in decimal degrees of the report.
- average_db: The average level of noise.
- timestamp: The UNIX timestamp (in seconds) of the report.
- duration: The duration of the sampling (in seconds).

Retrieving Last Noise Measurement (APIv2)

API endpoint: `[Event]/noise/last?originator=%s` (GET)

This endpoint returns exactly the same data as the Map endpoint, but just one item.

Retrieving Last Noise Measurement (APIv1)

API endpoint: `[Event]/noise/last/legacy?originator=%s` (GET)

This endpoint has the same functionality as in APIv2, but returns an XML object. For example:


```
<widenoisemap apiversion="2.0">
  <entry for="all">
    <lat>51.518144</lat>
    <lon>-0.127716</lon>
    <value>75.515205</value>
  </entry>
</widenoisemap>
```


This endpoint is required for some components of WideTag and hence just for compatibility purpose.

Get a Specific Noise Report

API endpoint: `[Event]/noise/[ID]` (GET)

This endpoint returns exactly the same data as the Map endpoint, but just one item, identified by a given ID. It is used to show a specific noise detection in a Twitter or Facebook message.

2.3.4 Air Endpoints

Requesting Map Data

API endpoint: `[Event]/air/?lat=%f&lon=%f&lat_delta=%f&lon_delta=%f &originator=%s` (GET)

This endpoint works similar to the noise map endpoint, but returns just those reports that have user_data_1 information in a slightly different structure. For example:


```
{
  "data": [
    {
      "id"="46047688470687",
      "geo_coord": [-122.406417, 37.785834],
      "user_data": "Text A",
      "avg_pollution":3.45,
      "timestamp":"1323033341"
    },
    {
      "id"="86543046464888",
      "geo_coord": [-122.406417, 37.785834],
      "user_data": "Text B",
      "avg_pollution":2.34,
      "timestamp":"1321006236"
    }
  ]
}
```


An empty portion of the map returns:


```
{
  "data": []
}
```


Additionally, it is possible to specify an originator to request just those reports from a particular device.

Retrieving Last Air Measurement

API endpoint: `[Event]/air/last?originator=%s` (GET)

This endpoint returns exactly the same data as the Map endpoint, but just one item.

2.3.5 Webapp Endpoints

The following sections cover REST endpoints providing data for the WideNoise web application. Their URLs all share a common prefix namely `event/{id}`. Thus, this prefix is omitted.

Set WideNoise id

API endpoint (private): `api/personal/set/widenoiseid/{wideNoiseId}` (POST)

This endpoint allows to set the users own WideNoise id. The path variable “wideNoiseId” specifies the id to set the users own WideNoise id to.

Response:

`Status code (401)`	if the user is not logged in
`Status code (409)`	if the WideNoise id is already taken
`Status code (200)`	if the WideNoise id was set sccuessfully

KML (Clusters)

API endpoint: `kml` (GET)

This endpoint allows to retrieve a KML (keyhole meta language) file aggregating the measurement data in clusters. There are several parameters to customize the returned data.

Parameters:

Name	Format	Default	Description
`bbox`	double,double,double,double	not required	Defines a bounding box to limit the returned data. If no bounding box (min_lon, min_lat, max_lon, max_lat) is given, all data is returned. If no bounding box is given no clustering occurs.
`dim`	int,int	not required	Specifies dimensions (in pixel) of the current map div element. Used for clustering. If no dimensions are given no clustering occurs.
`threshold`	int	`15`	Minimum distance (in pixels) a point must have from a cluster to not be added to it.
`personal`	boolean	`false`	Restricts the data to personal data (only works correctly when logged in).
`from`	YYYY-MM-DD_HH:mm:ss	minimum	Restricts to data recorded after the given data.
`until`	YYYY-MM-DD_HH:mm:ss	maximum	Restricts to data recorded before the given date.

Example:

```
kml?personal=true&dim=1900,671&bbox=-1.53,48.84,19.34,53.46
```

Result:


```
<kml><Document>
  <name>widenoise.kml</name>
  <open>1</open>
  <Placemark>
    <ExtendedData>
      <Data name="average_db"><value>76.50881954310344</value></Data>
      <Data name="p_artifical"><value>0.5043103448275862</value></Data>
      <Data name="p_disturbance"><value>0.5</value></Data>
      <Data name="p_feeling"><value>0.5</value></Data>
      <Data name="p_isolation"><value>0.5</value></Data>
      <Data name="youngest"><value>1341925189000</value></Data>
      <Data name="oldest"><value>1341831406000</value></Data>
      <Data name="count"><value>116</value></Data>
      <Data name="bounds"><value>
       [4.4025474, 51.185226, 4.4395266, 51.220554]</value></Data>
      <Data name="tags"><value>{"Trashtruck":1,"Outside":43,"Bar":2,"
       Motorcycle":1,"EvA":97,"Antwerpen":2}</value></Data>
    </ExtendedData>
    <Point>
      <coordinates>4.420051998275861,51.21157837931035</coordinates>
    </Point>
  </Placemark>

  ...

</Document></kml>
```


KML (Grid)

API endpoint: `kml/grid` (GET)

This endpoint allows to retrieve a KML (keyhole meta language) file aggregating the measurement data into grid cells. The cell size can be adjusted. The KML features 3D visualization of the grid cell's sample count.

Parameters:

Name	Format	Default	Description
dx	double	4	Grid width in degrees (longitude).
dy	double	4	Grid height in degrees (latitude).

Example:

```
kml/grid?dx=5&dy=7
```

Result:

```
<kml><Document>
  <name>widenoise.grid.kml</name>
  <open>1</open>
  <Placemark>
    <Style><PolyStyle><color>80ff9933</color></PolyStyle></Style>
    <ExtendedData>
      <Data name="average_db"><value>38.42845500000001</value></Data>
      <Data name="p_artifical"><value>0.5</value></Data>
      <Data name="p_disturbance"><value>0.5</value></Data>
      <Data name="p_feeling"><value>0.5</value></Data>
      <Data name="p_isolation"><value>0.5</value></Data>
      <Data name="youngest"><value>1334260520000</value></Data>
      <Data name="oldest"><value>1331348659000</value></Data>
      <Data name="count"><value>3</value></Data>
      <Data name="bounds"><value>
        [175.617, -40.39056, 175.64072,-40.35]</value></Data>
      <Data name="tags"><value>{}</value></Data>
    </ExtendedData>
    <Polygon>
      <extrude>1</extrude>
      <tessellate>1</tessellate>
      <altitudeMode>absolute</altitudeMode>
      <outerBoundaryIs><LinearRing>
        <coordinates>
          175.0,-42.0,30000.0
          175.0,-35.0,30000.0
          180.0,-35.0,30000.0
          180.0,-42.0,30000.0
          175.0,-42.0,30000.0</coordinates>
      </LinearRing></outerBoundaryIs>
    </Polygon>
  </Placemark>

    ...

</Document></kml>
```

Bounds

API endpoint: `kml/bounds` (GET)

This endpoints return the bounding box around all current samples or only around the users persinal samples.

Parameters:

Name	Format	Default	Description
`personal`	boolean	`false`	If set to `true`, the bounding box will be restricted to only personal data (only works correctly when logged in).

Example:

```
kml/bounds?personal=true
```

Result:

```
[4.406344,49.780712,9.974034,52.38942]
```

Measurement

API endpoint: `measurement/id` (GET)

This endpoint returns the measurement with the given a measurement “id” via path variable.

Example:

```
measurement/9094016899509432825
```

Result:


```
{
  "id":9094016899509432825,
  "client":"WideNoise/3.3.0 66 (Linux; U; Android 2.2.1; HTC Wildfire)",
  "event":"WideNoise",
  "ip":"82.44.210.137",
  "request_ts":1346413418000,
  "average_db":67.720306, "average_raw":0.024161,
  "device":"HTC Wildfire",
  "duration":5.0,
  "lat":51.488155,"lon":-0.30690053,
  "city":"Wealden","state":"England","country":"United Kingdom",
  "perception_feeling":0.8,"perception_disturbance":0.2,
  "perception_isolation":0.0,"perception_artificiality":1.0,"
  samples":"[
    \"0.16467011\",\"0.050304513\",\"0.022746556\",\"0.016479235\",
    \"0.014700989\",\"0.014404921\",\"0.014220835\",\"0.013906292\",
    \"0.013719226\",\"0.012825372\",\"0.012966252\"]",
  "tags":"[\"lawn  mower\"]",
  "sample_ts":1346413392000,
  "uid":"355797046081545",
  "map_data":"{
    \"geo_coord\":[-0.30690053,51.488155],
    \"average_db\":67.720306,
    \"timestamp\":1346413392,
    \"duration\":5}",
  "location_precision":"net", "location_accuracy":140.0,
  "user_estimate":0.0,
  "tagList":["lawn mower"]
}
```


Check for Update

API endpoint: `(personal/)hasBeenUpdated` (GET)

This endpoint allows to check if measurements have been recorded since a certain date. The corresponding URL can either be called with or without the prefix “personal”. When using the prefix only personal samples are considere. This only works when logged in.

Parameters:

Name	Format	Default	Description
since	YYYY-MM-DD_HH:mm:ss	required	Date to check for updates after.

Example:

```
hasBeenUpdated?since=2012-07-09_10:00:00
```

Response:

true	if a sample was recorded since the given date
false	if no sample was recorded since the given date

Average Decibel Value

API endpoint: `(personal/)averageDBValue` (GET)

This endpoints returns the average dB value for a given timespan. The data considered can be limited to personal data bys using the prefix “personal”. There also exist several short hand URLs for certain timespans, which are self-explaining:

- `averageDBValue/last/day`
- `averageDBValue/last/year`
- `averageDBValue/last/month`

Parameters:

Name	Format	Default	Description
`young`	YYYY-MM-DD_HH:mm:ss	required	Youngest sample to consider. Excludes the usage of “since".
`old`	YYYY-MM-DD_HH:mm:ss	required	Oldest sample to consider. Excludes the usage of “since".
`since`	YYYY-MM-DD_HH:mm:ss	required	Considering all samples since this date. Excludes the usage of “young" and “old”.

Example:

```
averageDBValue?since=2012-07-09_10:00:00
```

Result:

```
68.4195917758305
```

Latest Data

API endpoint: `(personal/)latestData` (GET)

This endpoint allows to retrieve the latest recorded data. When specifying the prefix “personal” this will be limited to personal data. Note that some parameters are only available for the non-personal URL.

Parameters:

Name	Format	Default	Description
`amount`	int	not required	Amount of latest samples to return.
`days`	int	not required	Limit to samples from last few days (only non-personal).
`filter`	boolean	not required	Filter samples with (0,0) coordinates are not considered (only non-personal).

Tags

API endpoint: `tags` (GET)

Allows to retrieve all tags in a certain area.

Parameters:

Name	Format	Default	Description
`bbox`	double,double,double,double	not required	Bounding box (min_lon, min_lat, max_lon, max_lat) to select tags from.
`limit`	int	not required	Limit the number of different tags.

Example:

```
tags?bbox=-0.364,51.45,-0.270,51.489
```

Result:


```
[
  {
    "latitude":51.46800457798164,
    "longitude":-0.3284920506605505,
    "latitudeTop":51.471535, "latitudeBottom":51.463463,
    "longitudeLeft":-0.3307655, "longitudeRight":-0.32164916,
    "name":"garden",
    "widenoiseId":"359918045471257",
    "event":"WideNoise",
    "count":545
  },

  ...
]
```


Continents

API endpoint: `continents` (GET)

Returns the number of measurements for different continents for the last few days.

Parameters:

Name	Format	Default	Description
`limit`	int	not required	Number of days to calculate the histogram from.
`array`	boolean	`false`	Specifies if the result will be a map of two arrays or a list of (continent, count) tuples.

Example:

```
continents?amout=3
```

Result:


```
{
  "Africa":317,
  "Asia":8483,
  "Australia":141,
  "Europe":15025,"
  North America":835,
  "South America":59
}
```


Coverage

API endpoint: `coverage/data` (GET)

Calculates the spatial coverage for each user of a designated area.

2012 © Copyright lies with the respective authors and their institutions

Parameters:

Name	Format	Default	Description
bbox	double,double,double,double	required	Bounding box (min_lon, min_lat, max_lon, max_lat) for coverage to apply to.
from	YYYY-MM-DD_HH:mm:ss	minimum	Restricts to data recorded after the given data.
until	YYYY-MM-DD_HH:mm:ss	maximum	Restricts to data recorded before the given date.
width	int	2000	The coverage alogrithm uses a grid internally. Specifies the grid's width.
height	int	2000	The coverage alogrithm uses a grid internally. Specifies the grid's height.
radius	int	10	For each sample the covergae algorithm draws a point to a grid. This specifies it's radius. The radius can either be in meters or on grid cells (default).
meters	int	false	Sets wether the radius is interpreted as grid cells (false) or as meters (true).
sortedByCoverage	boolean	true	The returned users can either be sorted by sample count or by coverage.
onlyRegistreredUsers	boolean	true	Sets if only statistics for registered users are returned.
overall	boolean	false	If set to true an overall coverage aggregating all users will be calculated.

Example:

```
coverage/data?bbox=-0.364,51.45,-0.270,51.489&from=2011-01-09_10:00:00
    &radius=200&meters=true&sortByCoverage=true&onlyRegisteredUsers=false
```

Result:


```
[
  {
    "a":"Not Registered",
    "b":{
      "coveredAreaPercentage":2.578125,
      "measuredGrids":103125.0,
      "count":77.0,
      "coveredArea":1704850.3223186063,
      "missingGrids":3896875.0
    }
  },{"
    "a":"Some Name",
    "b":{
      "coveredAreaPercentage":2.501625,
      "measuredGrids":100065.0,
      "count":80.0,
      "coveredArea":1654262.763663625,
      "missingGrids":3899935.0
    }
  },

  ...
]
```


2.4 Data Processor

All data that are sent to the REST server are written as they are to the database. The information, stored in the requests, are extracted from an independent data processor.

The data processor queries every three second for new data and tries to parse them. It is able to detect invalid data and can associate all data to their corresponding event. New and therefore unknown data object are treated as invalid and hence don't disturb the work of the data processor.

The data processor consists of four components as depicted in Figure 2.9. They are:

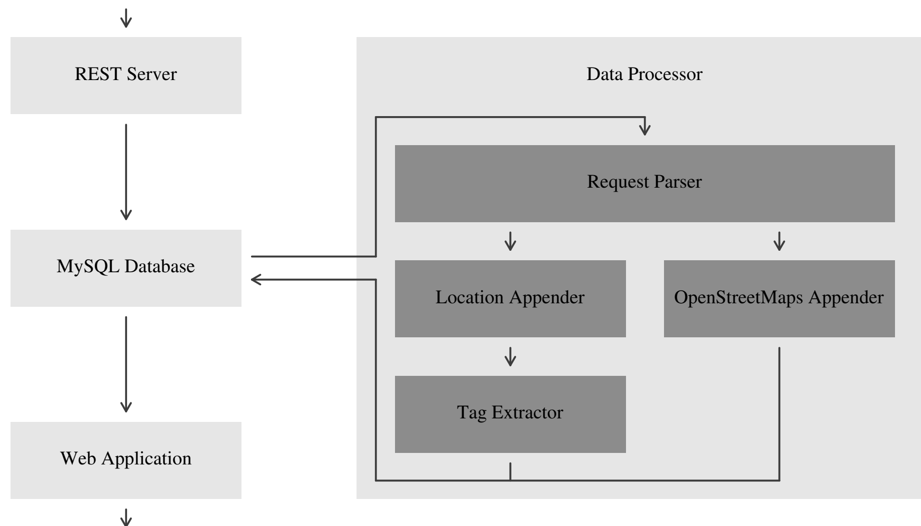


Figure 2.9: Component Overview of the Data Processor

1. Request Parser (see Section 2.4.1)
2. Location Appender (see Section 2.4.2)
3. OpenStreetMaps Appender (see Section 2.4.3)
4. Tag Extractor (see Section 2.4.4)

2.4.1 Request Parser

In the first step the data processor queries for new data from the measurements table identified by their event and format column's information. A new data set is identified by an empty state value (i.e., NULL). All data that has already been handled by the data processor, successful or not, have a state value different from NULL. Successful ones are identified by a positive value and invalid ones by a negative value.

First, the column id and the timestamp are extracted from these data, regardless of their type. Every report data will get the same id as the corresponding raw dataset in the measurements table. The timestamp is extracted since it is needed to assign malformed tag request to their corresponding reports (more on this bellow).

The type of the current raw data is determined in the next step. In order to archive this, the data processor checks the JSON object from the rawdata column for the following information:

WideNoise report request : Does the JSON object contain the fields average_raw, average_db, device, duration,geo_coordinate, perceptions, samples, timestamp, and hash? Does the geo_coordinate JSON array contain exactly two values (latitude and longitude)? Does the perceptions JSON object contain the fields artificiality, disturbance, feeling, and isolation)?

WideNoise tag request : Do the JSON object contain the fields timestamp, tags, and hash? Does the JSON tags array contain at least one value?

WideNoise Tweet request : Does the JSON object contain the fields uid, device, statuses, and hash?

Air Quality App report request : Does the JSON object contain the fields co_1, co_2, co_3, co_4, device, geo_coord, hum, no2_1, no2_2, o3_1, temp, timestamp, user_data_1, voc_1,

and hash? Does the geo_coord JSON array contain exactly two values (latitude and longitude)?

The data set is marked as invalid if the current JSON in the rawdata column didn't match any of the patterns above. In such a case, the state value of the corresponding measurements entry is set to "-1".

The rawdata is parsed based on the type that was detected by the patterns above. Depending on the type the following actions are performed in addition to a mere extraction of data:

WideNoise report request : The data processor changes the timestamp to that one recorded by our REST server if it lies behind that one. Since there was a type in the first WideNoise iOS version that transmitted the location accuracy, it checks for the field location_accurancy if there is no value for location_accuracy, when the request was received by our REST server.

WideNoise tag request : Since the WideNoise Android version that we get from WideTag had the bug that it didn't contain the information to which report it belongs, the data processor checks for the existence of the id field. It queries for the last report request received from the current device (identified by its uid) and uses that id value. Then it removed all leading and trailing whitespaces from every tag contained in the JSON tag array.

WideNoise Tweet request : Since there was a bug in the early version of the Tweet request the data processor replaces every occurrence of a double quote sign from the contained Twitter status messages with a single quoute sign. This is necessary since the double quote sign is used by JSON as separator and would obstruct the JSON parser from parsing the JSON object.

Air Quality App report request : Since the Air Quality App is still in a very early stage, there are no special treatments for the corresponding data.

The IP source address is extracted from the HTTP header of the header column if the ip column contains the IP address of our proxy server. This is currently everytime the case since the EveryAware server is located in our internal network. The IP information is found in the x-forwarded-for header field if present. It is set to NULL if there is no such header or it contains the value unknown.

Afterwards the information from the client and event column are copied to the parsed object. The parsing of one measurement is finished by storing it in the database an mark the corresponding measurements entry as processed successfully (i.e., the state value is set to "1").

2.4.2 Location Appender

Since a lot of measurements didn't contain any location information, we decided to evaluate the source IP address of the measurements. Reports without a real location information are identified by their lat and lon content which has to be zero as well as the content of location_precision which should be "dev" (the default value).

To retrieve additional location information, we query the free IP address geo-location tool IPInfoDB. If IPInfoDB responds with a location, the data processor writes this information to the corresponding report and changes the value of location_precision to "ip".

If we didn't receive any location information, the data processor writes "none" in the location_precision column.

2.4.3 OpenStreetMaps Appender

In order to enrich the reports with additional information, the data processor queries OpenStreetMaps[25] for more location information about the stored location (identified by its latitude and longitude value). The data processor writes the received city, state, and country name to the corresponding report columns.

2.4.4 Tag Extractor

In the last stage, the data processor extracts the tag information of every report into a separate table called tags. This information is used by the map view on the EveryAware WideNoise web page. Every tag data set is enriched with the id of the source report, the geo-location (i.e., latitude and longitude), the name of the event during that the tag was captured, the uid of the source device, and the Ubicon account user name, if existent.

2.5 Ubicon

Ubicon is a web framework supporting event and user management simultaneously. The following sections give an overview of about its architecture and describe the event category WideNoise. WideNoise features access to and visualization of noise samples recorded by individual users.

2.5.1 Overview

As mentioned before, the Ubicon framework supports event as well as user management. This section will first introduce the event management concept and then describe how the user management is designed.

Event Management

Events are the basic building blocks of the Ubicon framework. Each event is an instantiation of an event category. An event is identified by an unique id. The id is used to define the event's URL. All pages belonging to the same event share the same prefix:

`/event/{id}/**.`

This allows to run several instances of the same event category on the same server. While the event type is the same, the content can be different. The functionality is shared. For example, if one event category is providing a system for managing a conference, each instantiation is associated with a single conference. Or if one event category is taking care of providing a data collection framework for certain events like a football game or a rock concert, the different instantiations can cleanly separate the collected data. An event instance is instantiated by an entry in the data base. Section 2.5.2 gives a detailed example of an event category called "noise", which provides functionality for collecting and displaying noise data.

User Management

The user management is an integral part of Ubicon. It has been designed to support a consistent user experience spanning the different events the Ubicon framework can host. First and foremost it provides basic functionality including

[25] http://wiki.openstreetmap.org/wiki/Nominatim

- user registration
- user login
- password reset and
- editable user details.

These features are available through any event hosted by the Ubicon framework. While they share the same functionality the user interface can be adjusted for each event category individually.

2.5.2 WideNoise

The “noise” event category provides functionality to aggregate, summarize and display noise related data collected by the WideNoise smartphone application (see D1.1 and D3.1). It provides several static pages like the front page and the about pages. Figure 2.10 shows screenshots of the front and about pages. For the remainder of this section we will talk about the “noise” category instance “widenoise”.
The front page of the event “widenoise" is accessible using the URL

`/event/widenoise.`

This is also the common prefix for all further URLs concerning the “widenoise" event. All other URLs in this section will relative to this prefix.
The front page provides access to all important functionality of and information about the “widenoise” event and allows sharing the event on Twitter or Facebook as is illustrated in Figure 2.11.
The about page is accessible using the URL

`about.`

It describes the project in detail and provides support for example by answering frequently asked questions.
The remainder of this section covers the statistics displayed on the front page as well as personal statistics and data accessible by the user. Afterwards a page is introduced summarizing the spatial coverage concerning noise samples of a designated area. Then the map page is introduced which displays noise data as well as other collected data. Finally one subsection explains how users are identified using the so called “WideNoise id”.

Statistics

Several statistics have been implemented. Some of the are shown directly on the front page. Some are only visible to the user personally. Others are implemented using separate pages like the spatial coverage for a designated area. These statistics are described in detail in this section.

Front Page Statistics The front page provides several summarizing views on the data. These views use data provided by the REST server. Below the views are listed with reference to the URLs they are using. For further details on the URLs please refer to Section 2.3.5.

- A histogram summarizing the number of measurements for each continent for the last three days (see Figure 2.12(a)). The histogram is calculated using data from the API URL

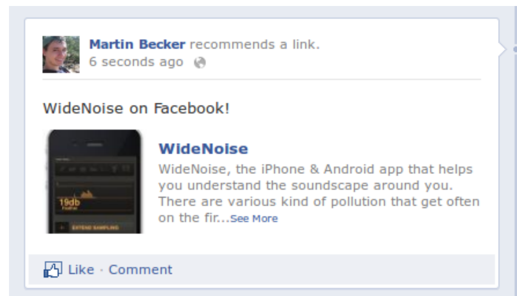
  `continents.`

(a) Front page.

(b) About page.

Figure 2.10: Static pages of WideNoise event of the “noise” category.

(a) Sharing functionality on front page.



(b) Shared content on Facebook.

(c) Shared content on Twitter.

Figure 2.11: Social sharing functionality on the “WideNoise” event’s front page.

- A table listing the registered users with the most samples overall and a table listing the registered users with the most samples covering the last two months (see Figure 2.12(b)). The user statistics are not accessible using a REST interface as of yet.
- A table showing latest recordings and a table with average values for the last day, month and year (see Figure 2.12(c)). The latest recordings can be clicked on leading to the map view focusing on that specific sample. The users’ latest recordings and the average values can be accessed using the data from

  `latestData`

  and

  `averageDBValue`.
- A scatter plot showing the dB values for the measurements spanning the last three days (see Figure 2.12(d)). The scatter plot is parsed from the same data as the table showing latest recordings.

Personal Statistics The personal statistics can be accessed by the user by referring to the URL

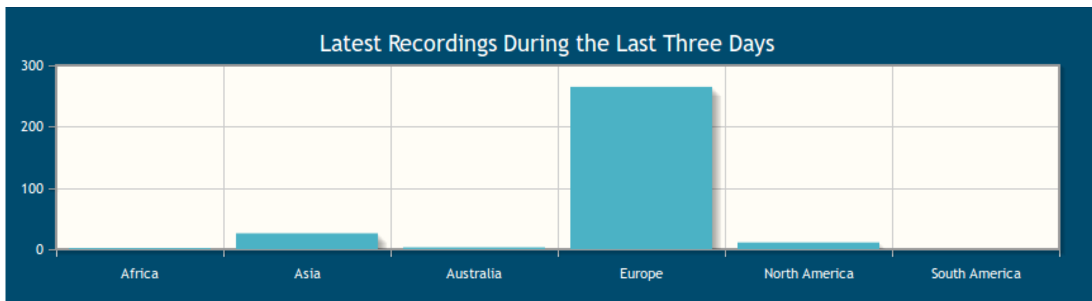
`personal`.

A screenshot of this page is shown in Figure 2.13.

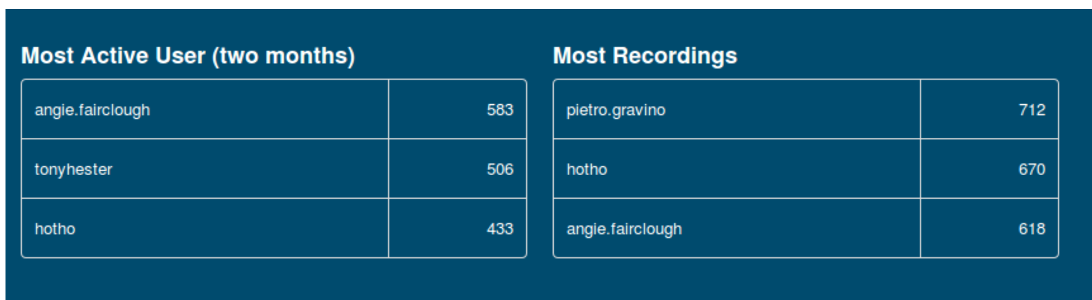
This page displays the latest recordings of the user as well as a scatterplot of her latest measurements and provides a KML file export (Keyhole Markup Language) containing all measurements of that user to display for example in Google Earth (see “Download KML" in Figure 2.13). This data can also be accessed using the REST server using the URL

`kml?personal=true`

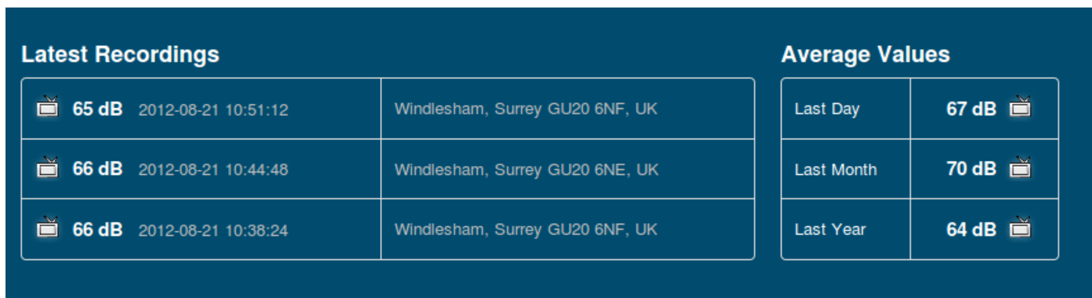
Again, see Section 2.3.5 for reference.



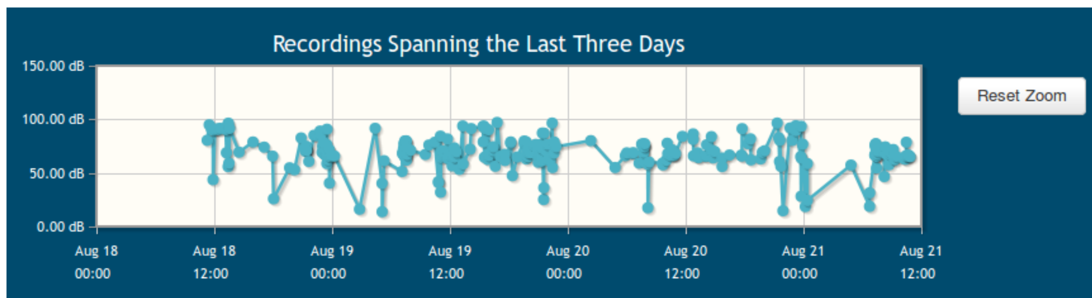
(a) Histogram showing measurment count by continent spanning the last three days.



(b) Most active users.



(c) Latest recordings and average decibel values.



(d) Scatter plot of measurements.

Figure 2.12: Front page statistics for the WideNoise event of the “noise” category.

2012 © Copyright lies with the respective authors and their institutions

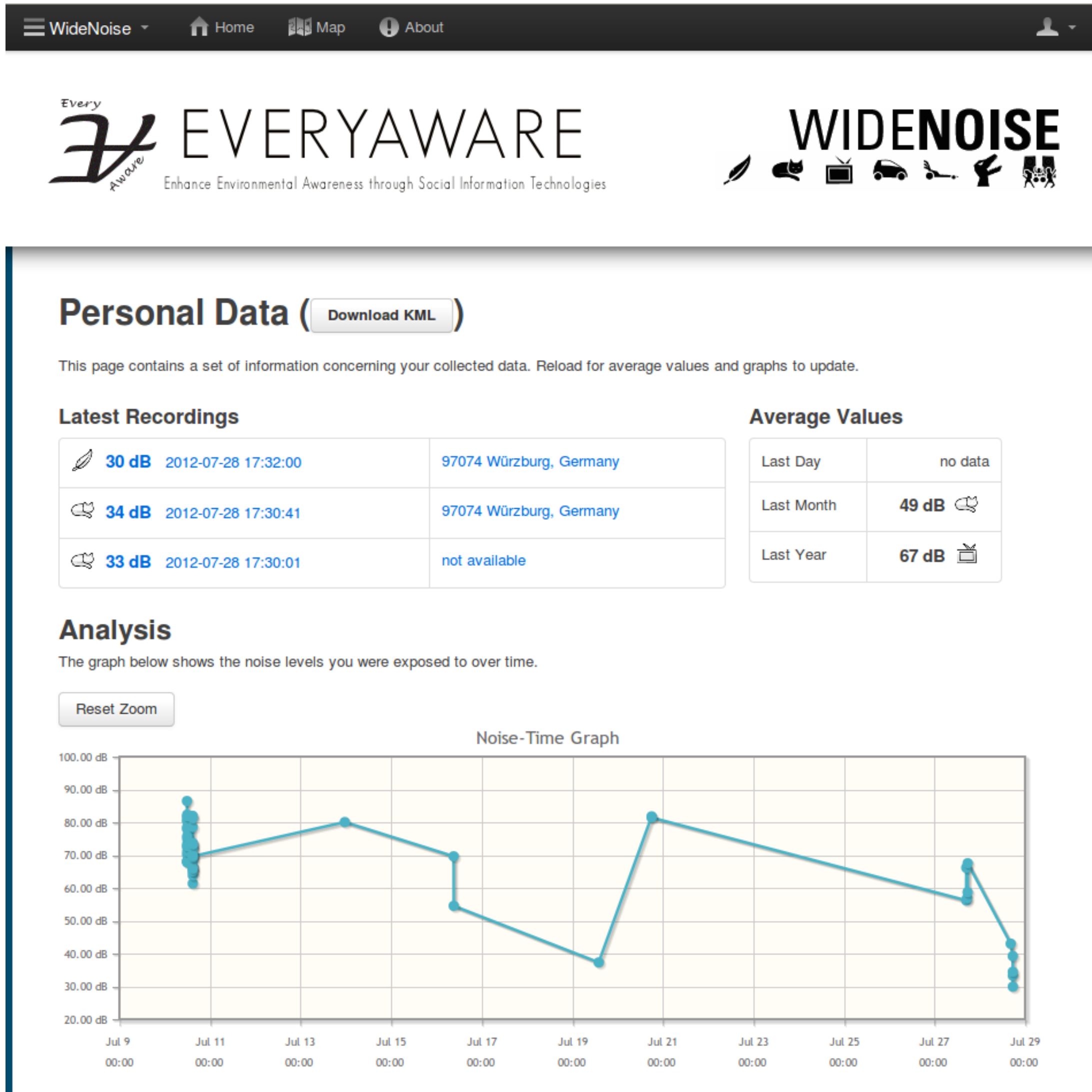


Figure 2.13: User statistics for the WideNoise event of the “noise” category.

Table 2.1: Coverage page URL parameters.

Name	Format	Default	Description
`bbox`	double,double,double,double	required	Bounding box (min_lon, min_lat, max_lon, max_lat) specifying the area to to calculate the coverage of. A corresponding rectangle is shown on the map on the coverage page.
`from`	YYYY-MM-DD_HH:mm:ss	not required	Specifying the first measurement to consider for the coverage. This date is displayed.
`until`	YYYY-MM-DD_HH:mm:ss	not required	Specifying the latest measurement to consider for the coverage. This date is displayed.
`reload`	int	no reload	Time interval to reload coverage data.
`style`	String	not required	Can be set to “rome” or “fullscreen” for different page styles.
`showCoverage`	boolean	no reload	only applicable to style “fullscreen”. This will trigger if explicit coverage information is shown or not.

Coverage The coverage page is accessible using the URL

`coverage`.

Figure 2.14 shows a screenshot of the coverage page. The coverage page allows to designate an area and a timespan as well as a perimeter covered by each measurement. It then computes the spatial coverage for each user in the designated area. Besides calculating the coverage for registered users only, the coverage page can be configured to show the coverage of not registered users as well. They will show up with the name “Not Registered”. Several designs of the page are available. The exact URL parameters to customize this page are shown in Table 2.1. The table only includes those parameters specific to the visualization. The data specific parameters are described in Section 2.3.5 for the URL

`coverage/data`.

The algorithm used to calculate the coverage is grid based. An equally spaced grid is placed over the designated area. Each grid cells corresponds to a pixel of a binary image. Each measurement is drawn as a circle with the radius derived from the specified perimeter around the measurement. The coverage is calculated from comparing the amount of painted pixels and their corresponding area against those pixels which have not been painted.

Map

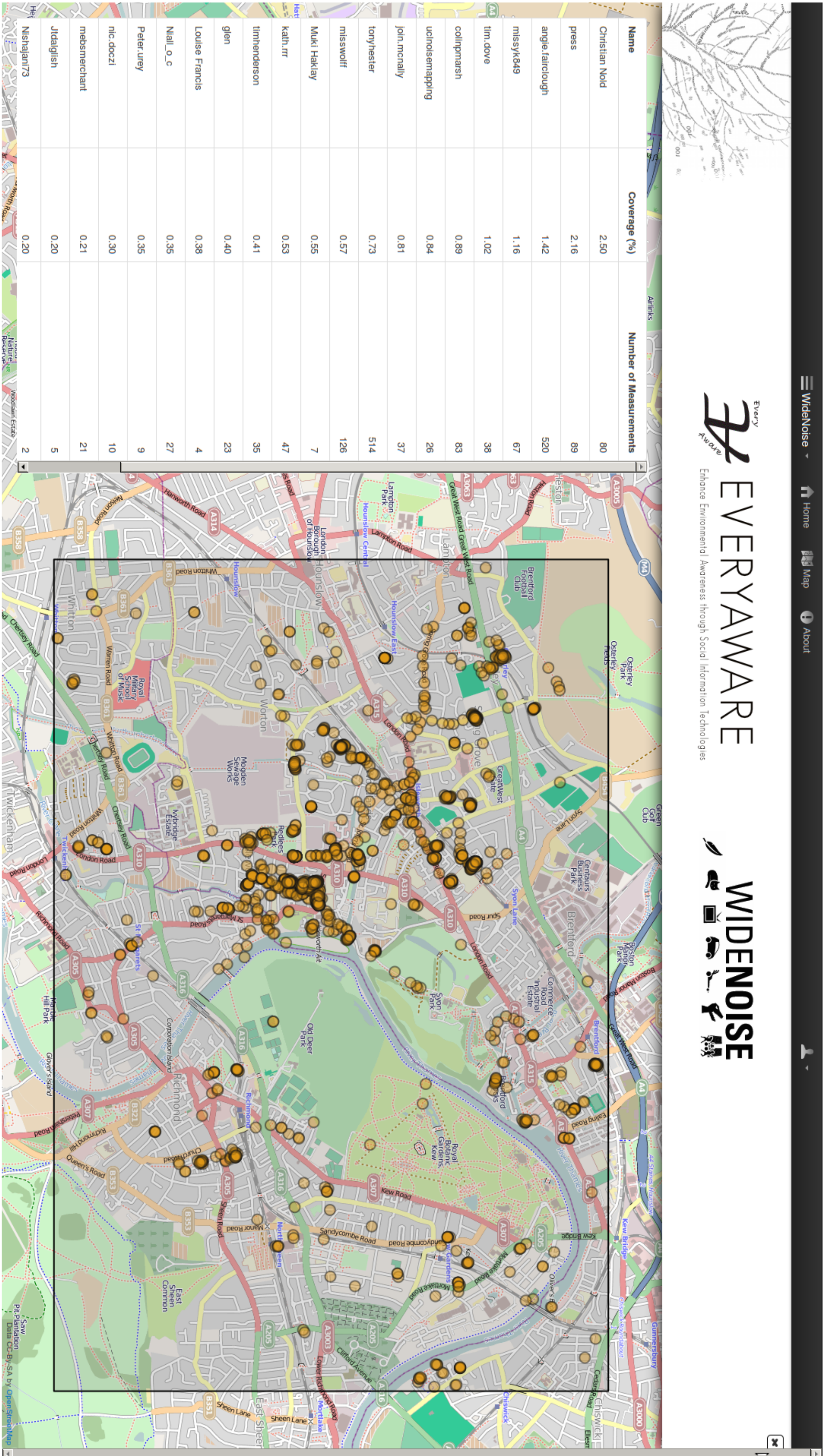
The map page is available by accessing the URL

`/event/widenoise/map`.

Figure 2.15 shows a screenshot of the map page. It shows an aggregated view on all the noise measurements. Using the default view, each marker on the map represents a cluster of noise measurements. The color visualizes the average decibel value of the cluster and the number on each marker represents the number of measurements the cluster contains. When clicking on a cluster a popup is showing more details about the cluster including

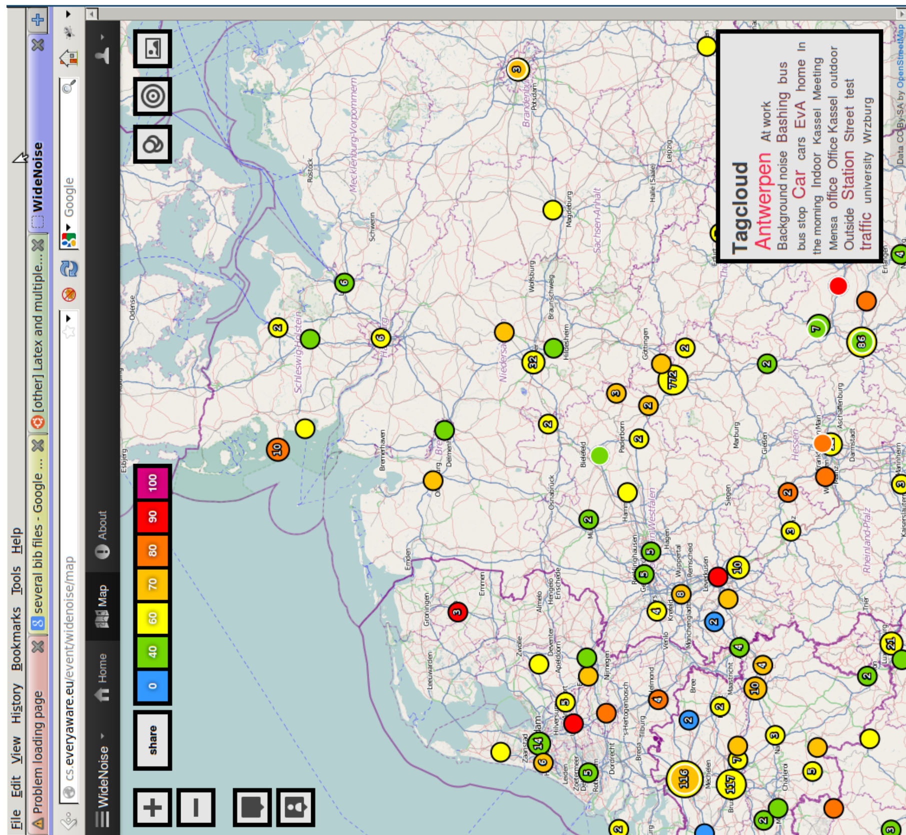
- average dB value,
- number of recordings,
- youngest and oldest sample,
- perception values as recorded by the WideNoise smartphone application (see Deliverable 3.1) and
- a list of tags assigned to samples contained by the cluster.

2012 © Copyright lies with the respective authors and their institutions

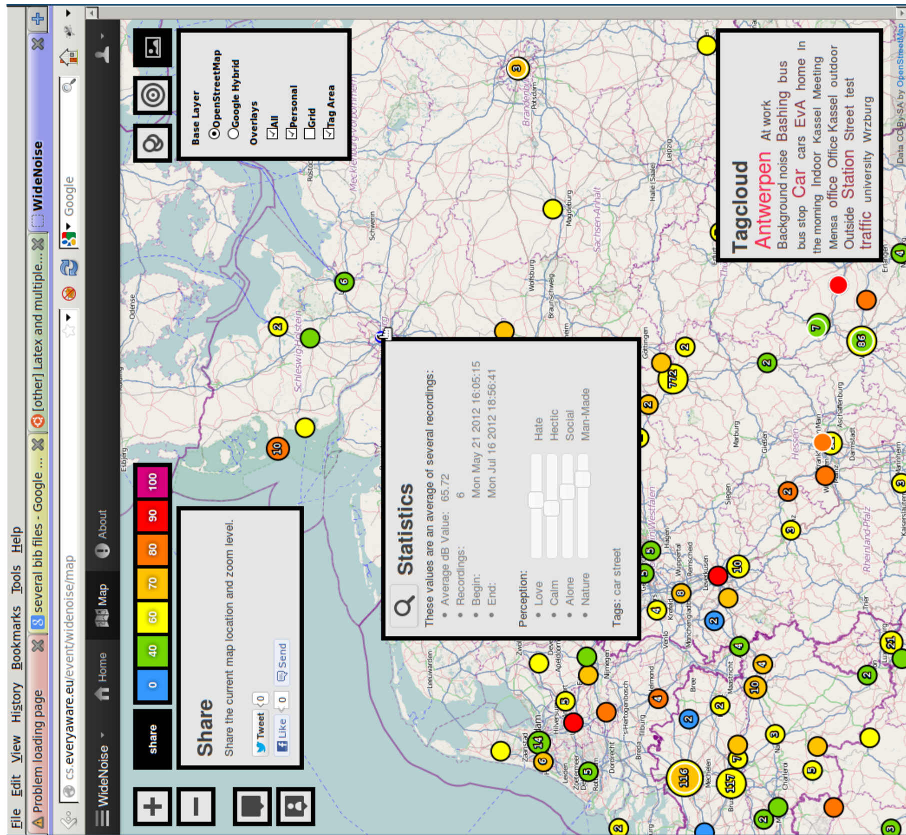


Name	Coverage (%)	Number of Measurements
Christian Nold	2.50	80
press	2.16	89
angie.fairclough	1.42	520
missyk849	1.16	67
tim.dove	1.02	38
colinpmarsh	0.89	83
uclnoisemapping	0.84	26
join.mcnally	0.81	37
tonyhester	0.73	514
misswolff	0.57	126
Muki Haklay	0.55	7
kath.rrr	0.53	47
timhenderson	0.41	35
glen	0.40	23
Louise Francis	0.38	4
Niall_o_c	0.35	27
Peter.urey	0.35	9
nic.doczi	0.30	10
mebsmerchant	0.21	21
Jtdalglish	0.20	5
Nishajani73	0.20	2

Figure 2.14: Coverage page of the “WideNoise” event.

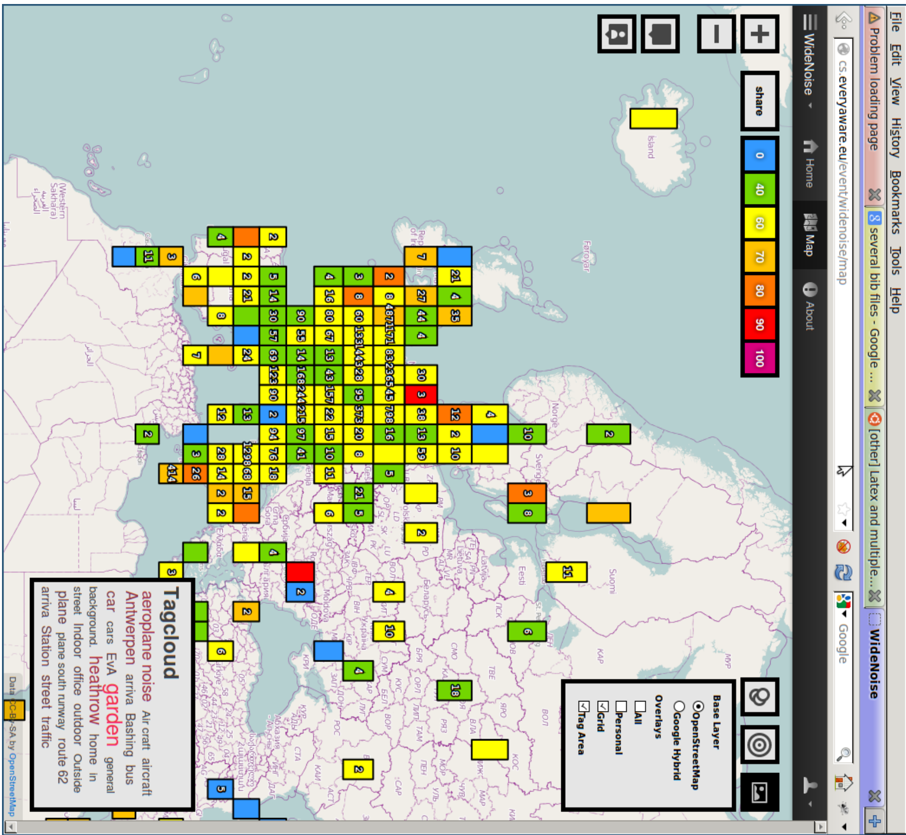


(a) Default view.

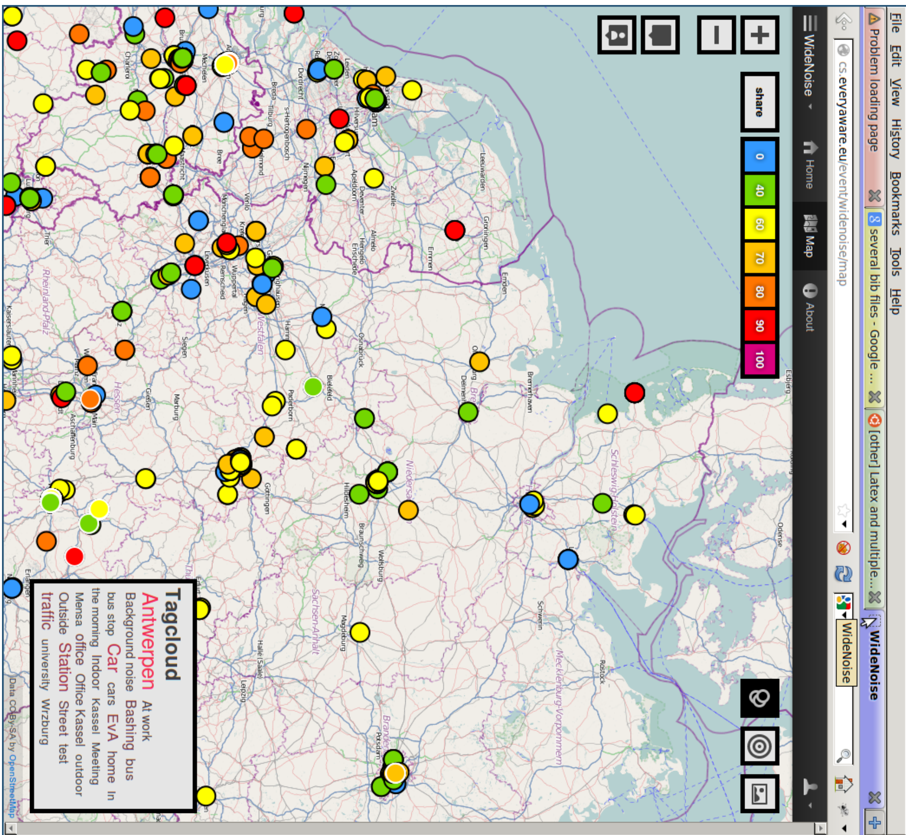


(b) Controls expanded and displaying aggregated cluster information.

Figure 2.15: Map page of the “WideNoise” event.

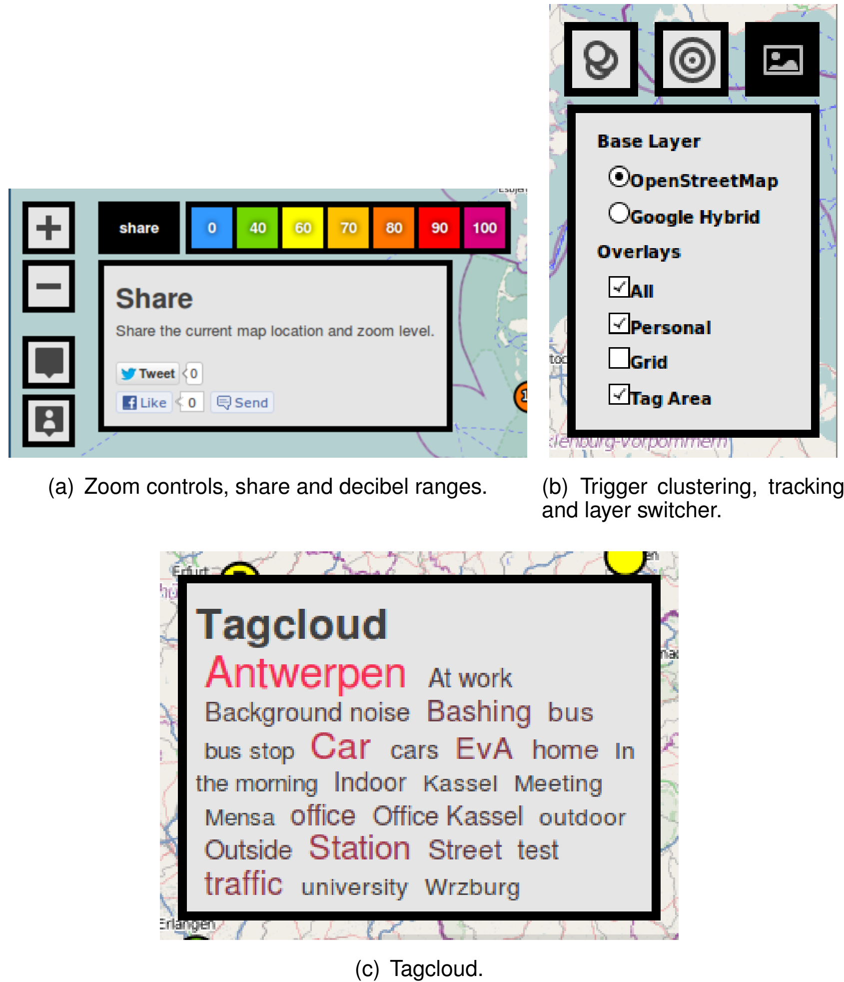


(a) Grid view.



(b) No clusters.

Figure 2.16: Data representations on the map page of the "WideNoise" event.



(a) Zoom controls, share and decibel ranges.

(b) Trigger clustering, tracking and layer switcher.

(c) Tagcloud.

Figure 2.17: Elements on the map page of the “WideNoise” event.

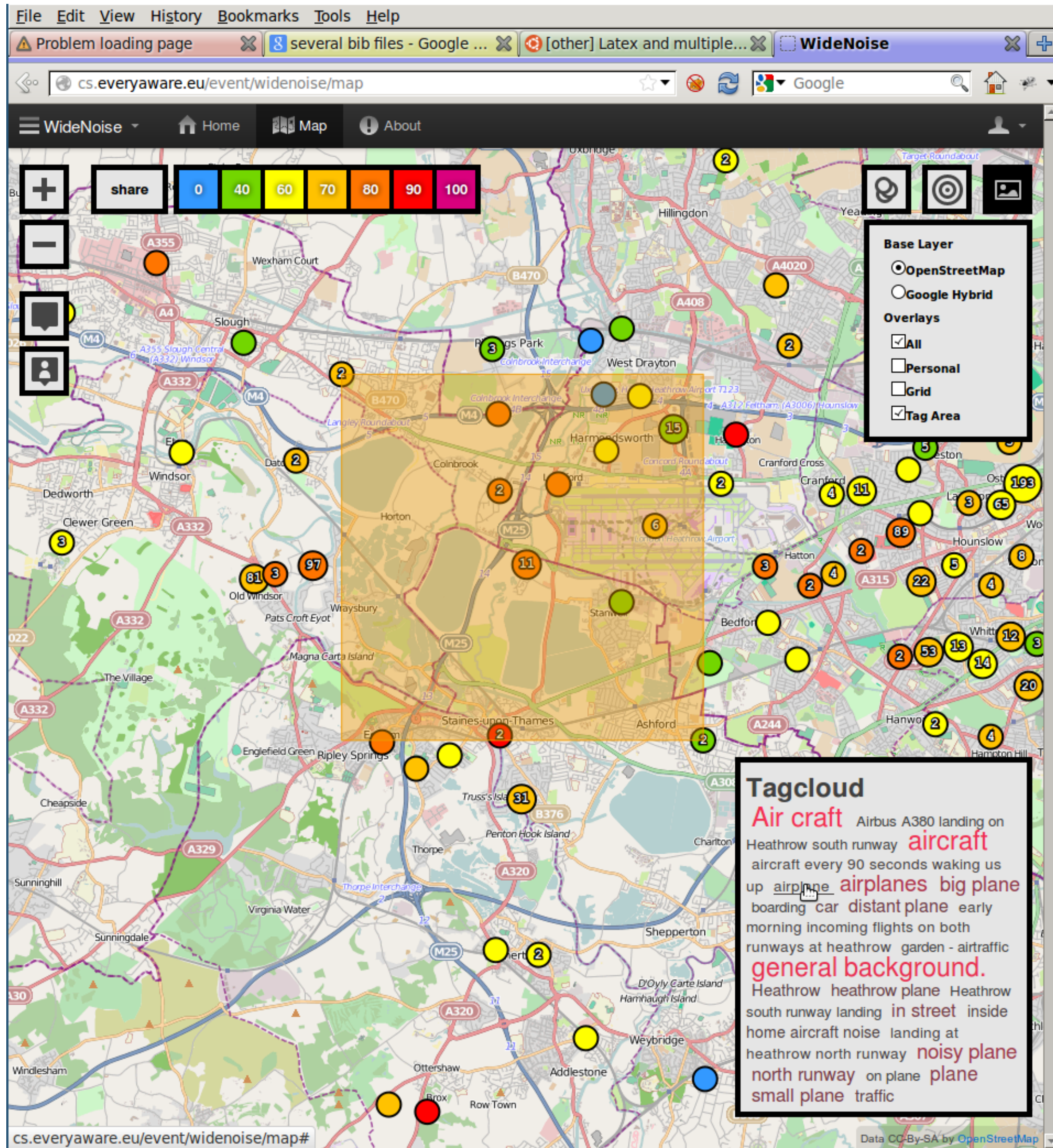


Figure 2.18: Tagcloud functionality on the map page of the “WideNoise” event..

Table 2.2: URL parameters for the map page of the “WideNoise” event.

Name	Format	Default	Description
`id`	long	not required	Tries to retrieve a certain measurement by its id and zooms in on that sample opening a popup for additional information.
`lon`	double	not required	Specifies a longitude value to focus on. Can only be used in combination with “lat".
`lon`	double	not required	Specifies a latitude value to focus on. Can only be used in combination with “lon".
`zoom`	double	not required	Specifies an initial zoom level. Can only be used in combination with “lon" and “lat".
`promotion`	boolean	`false`	If “promotion” is set to `true` the record tracking feature is enabled by default and at the bottom left of the screen three QR-Codes are displayed one leading directly to the map page and the other two linking to the WideNoise smartphone applications on Apple and Android.

The top left corner of the map view shows several controls that allow zooming. The plus and minus signs allow to zoom in and out. The buttons below that are used to either zoom on all the data on the world or to focus and zoom in only on personal data. The color scale illustrates which decibel values are mapped to which color. The “share” button provides functionality to share the current view on the map on Twitter or Facebook.

The top right corner of the map shows three buttons. The rightmost one provides a layer switcher which allows to select an OpenStreetMap[26] base layer or a base layer provided by Google[27] (satellite view). There are four overlays to choose from:

- “All”, which shows all the data.
- “Grid”, which clusters measurements using a grid.
- “Tag Area”, which shows the tag area when a tag is hovered in the tagcloud box
- Personal (only when logged in), which shows only the personal samples. Personal samples are displayed with a white border instead of a black one.

The middle button allows to track measurements as they come in. When a sample is recorded the map focuses on that measurement and automatically displays the corresponding popup. The leftmost button allows to disable clustering. The measurements are shown stacked upon each other.

At the right bottom a tagcloud is displayed which shows the tags assigned to samples in the currently viewed section of the world map. If a tag is hovered an area is highlighted, where this tag occurs. When the hovered tag is clicked the map zooms into the highlighted area.

The map view also supports several URL parameters. Those are described in Table 2.2.

The map page is implemented using OpenLayers[28] and displays KML files. These KML files can be accessed using the URL

```
kml
```

or

```
kml?personal=true
```

for personal data.

[26] `http://www.openstreetmap.org`
[27] `http://maps.google.de`
[28] `http://openlayers.org/`

2012 © Copyright lies with the respective authors and their institutions

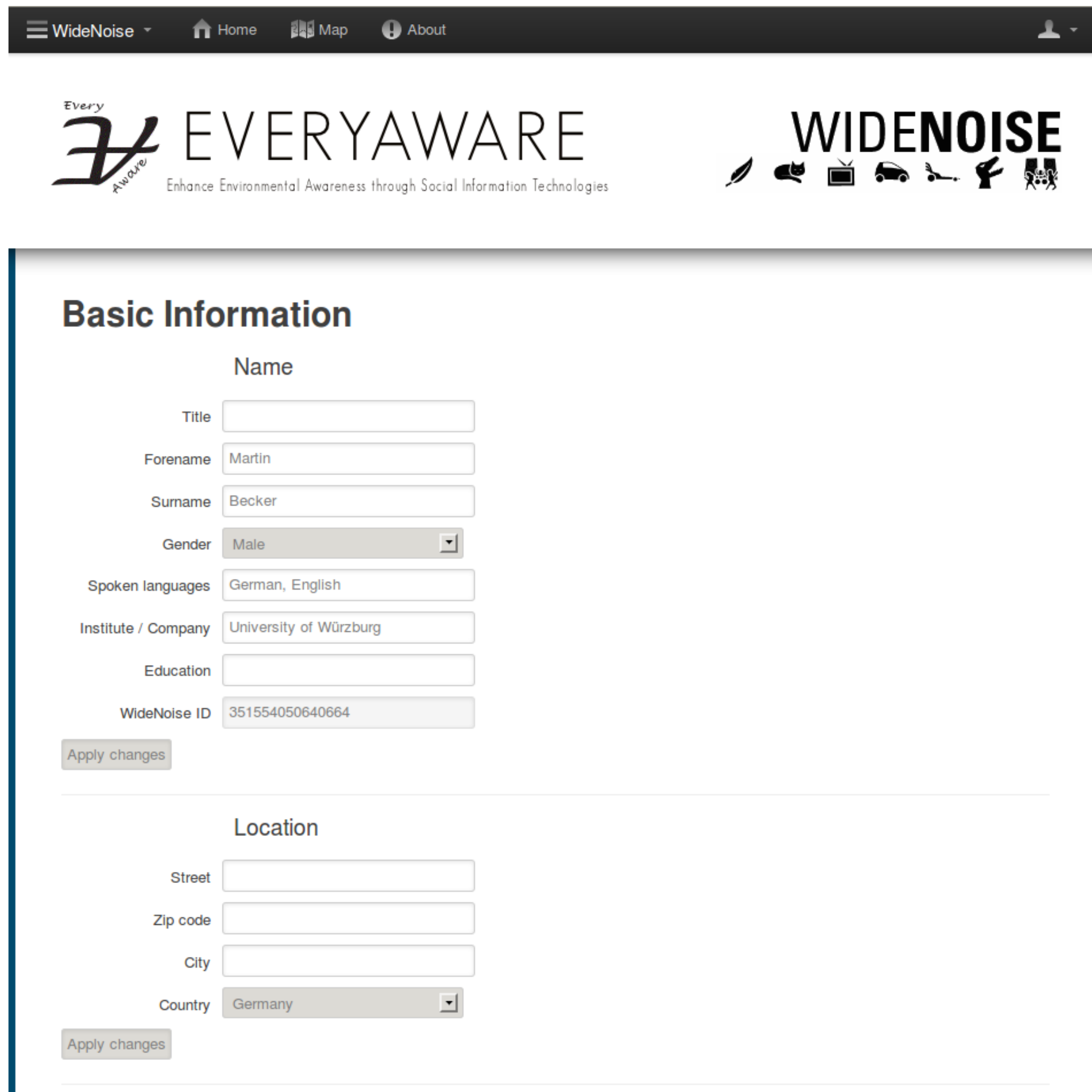


Figure 2.19: Profile page.

User

As mentioned in the sections about personal statistics and the map page the user has several possibilities to access personal data (see Figures 2.13 and Figures 2.15). The user can also enter a set of personal information on the profile page. The profile page can be accessed using the URL

`settings`.

Figure 2.19 shows part of the profile page.

In order so associate a user account with the data collected from a smartphone, the data is always sent including a so called “WideNoise id”. A WideNoise id is the unique device id derived from the cell phone. On Android phones this id is the IMEI (international mobile equipment identity) for GSM (Global System for Mobile communication), the MEID (Mobile Equipment Identifier) or ESN (Electronic Serial Number) for CDMA (code Division Multiple Access).

Each user can connect to a single WideNoise id. This is possible by logging in or registering an account from the cellphone. The dialog allows to backup the old WideNoise id and set the new one.

Chapter 3

Gaming Platform

3.1 Introduction

In the EveryAware project subjective data, i.e. information about what people think or how people feel, have a central role. Through the analysis of this sort of information we expect to understand the awareness dynamics which will eventually lead to behavioral shifts. To gather subjective data can be a treacherous task because they can be obtained, by definition, only by humans. There are several strategies to get these information. The most common approach is data mining, for example by performing a web-crawling of a social network. In this way, people opinions are gathered together with a lot of less useful information, and it is difficult to isolate the interesting part. Beside this, the approach is implicitly “passive”, let us say just observational, so it may be subject to a lot uncontrolled, and often unknown, biases. A more direct approach is needed, in order to get a better kind of subjective data.

The use of web-based games [von Ahn, 2006] for research purposes is a fast spreading phenomenon, changing the way research activities are conducted and how data are generated in many scientific fields. Two paradigmatic examples are *Foldit*[1] [Cooper et al., 2010], a game in which players are challenged to guess the 3D structure of a protein, and *Planet Hunters*[2] [Fischer and et al., 2011], by which participants can help in identifying new extra-solar planets using NASA data of star brightness.

The above mentioned projects have in common the involvement of individual volunteers or networks of volunteers, many of whom may have non specific scientific training, to perform or manage research related tasks in scientific projects. In this sense there are two examples of *citizen science* [Arnstein, 1969; Goodchild, 2007; Paulos et al., 2009], i.e., a long-standing series of programs traditionally employing volunteer monitoring for natural resource management. In recent years, also thanks to the Web 2.0 explosion, citizen science projects are becoming increasingly focused on scientific research [Nosek et al., 2002; Salganik and Watts, 2009] and amazing results have already been obtained. For example, the 3D structure of viral enzymes that challenged scientists for years has been discovered thanks to the efforts of Foldit players [Khatib and et. al, 2011], and new candidate planets identified by Planet Hunters' players managed to survive data verification tests [Fischer and et al., 2011]. It is worth to note how, in both cases, the engaging graphic interfaces boosted the participation quantitatively and qualitatively. In particular, in the Foldit case the game motivation has also proved to be a fundamental ingredient for web-based experiments realization, making the experiment results more numerous and reliable and, at the same time, less expensive than in other kind of experiment.

In a parallel development, the idea of *crowdsourcing* is at the heart of online labor markets such as Amazon Mechanical Turk (AMT), where a job is distributed by employers in small sub-tasks

[1] http://fold.it
[2] http://www.planethunters.org

2012 © Copyright lies with the respective authors and their institutions

that workers can perform. Interestingly, AMT has proven to be useful also for scientific purposes [Chilton et al., 2009; M and JW, 2009; Paolacci et al., 2010], e.g., as a powerful tool for recruiting experimental subjects and facilitating their reward through monetary payoffs. Early experience with crowdsourced experiments has paved the way to the recognition that web experiments, even despite having a partial control of the way participants are recruited and of the context in which tasks are executed, can be successfully used to study human collective behavior and cognition. The blog `http://experimentalturk.wordpress.com/` presents an early review of existing replications on AMT of classic experiments on individual and interactive decision making, and provides first elements of validation of experimental practices in the web [Siddharth and Duncan, 2011]. AMT has also opened the door for exploration of processes that outsource computation to humans on a large scale. These human computation processes hold tremendous potential to solve a variety of problems in novel and interesting ways. Human ability to easily solve tasks which are difficult to solve by setting up efficient algorithms has been largely exploited for instance by Google in labeling images (through the ESP collaborative game [von Ahn and Dabbish, 2004]), language translators, etc. Although the tenets of human computing are being increasingly exploited, its use in the scientific community still lacks of systematization. The realization of a single project often requires substantial effort and web-based experiments are still far from being standard research tools. The lack of tools that can greatly simplify and standardize the design of web games and experiments is a major bottleneck in the exploitation of such new research opportunities. Despite its versatility, [Paolacci et al., 2010] AMT has not been conceived as a platform for experiments. AMT itself offers, as other web sites do, some visual tools to develop simple interfaces (e.g. for polls) but it lacks infrastructures for the realization of games, which require more complex interactions, and thus more complex interfaces. Moreover AMT is based on economic motivations.

Experimentalists are left with the task of designing their own software solutions to manage interactions among participants and to build effective interfaces. Moreover, individual solutions to such problems often remain insulated with little or no cumulative growth of tools and solutions. This is the reason why it is important to develop a versatile platform to implement social *games*, to take advantage of the game motivation. This is the aim of Experimental Tribe, or XTribe, the word "game" being intended as an interaction protocol among a few players implementing a specific task as well as a synonym of experiment on interactive behavior.

In the EveryAware project, the role of the Experimental Tribe platform will be to make easy to set up "games" with the purpose of gathering subjective data about environmental issues, urban strategies and other topics related to the project.

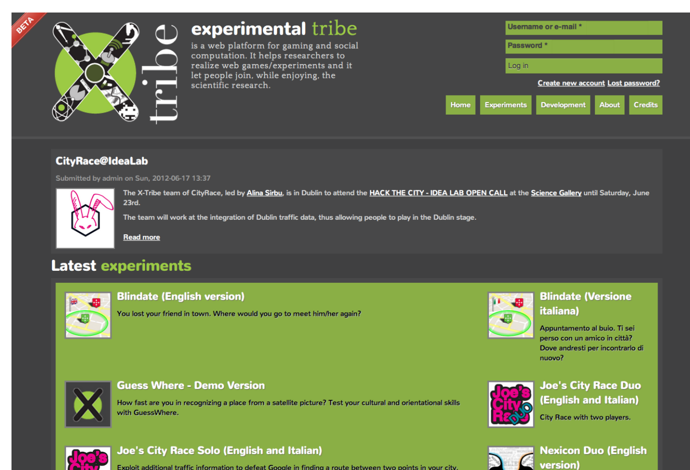


Figure 3.1: A screenshot of the XTribe homepage.

3.2 Experimental Tribe

Experimental Tribe (ET) is a platform for web-based experiments and social computation. It is currently available in beta version at www.xtribe.eu as showed in Fig. 3.1. ET is aimed at both gathering otherwise separate efforts to use web resources for scientific purposes and at providing the community with a tool to design experiments on the web, bypassing much of the “hard work”. The benefit is twofold: on the one hand, it allows virtually any researcher to realize his own experiment with minimal effort, paving the way of the use of the web as a standard “laboratory” to perform experiments. On the other hand, it can be a strong “basin of attraction” for people willing to participate to experiments, making in this way recruitment much more easier than for single-experiment platforms.

The idea behind ET is quite simple. When implementing a web experiment there are a lot of parts which have almost nothing to deal with the experiment itself and are common to almost all the kinds of web experiment, such users handling, interface hosting, security and privacy issues, etc. ET can take care of all these aspect allowing researchers to focus only on the interface and the logic part of the experiment. Thanks to ET it is possible to implement a simple multi-player web-game, with a users registry, a rank system and a guide page in few hours.

There are several active games on the platform. The most interesting in the context of the EveryAware project are *Blindate* and *Joe’s City Race*.

3.2.1 Blindate

Blindate is a collaborative game, very close to the well known *Schelling’s Games* first introduced in the early ’60s [Schelling, 1960]. In Schelling’s original version (one of many similar problems), two players, unable to communicate with each other, were asked to find a point on a map where to meet, i.e. they had to find a strategically salient “focal point” among a potential infinity of solutions to the coordination problem. Since Schelling’s seminal contribution, many versions of “Schelling games” have been used to investigate strategic salience, i.e. the individual ability to guess recursively what the other guesses that he will guess is salient, an so forth [Crawford et al., 2007; Mehta et al., 1994].

The game

In our custom version, two players, again unable to communicate to each other, are shown a portion of the map of a real city and are asked to point to a location in a given area where they think it is more likely to meet each other. The reward is a score depending inversely on the distance between the guesses. They can guess for a maximum of 5 times if their guesses do not match. In addition, after the choice, participants may optionally explain with suitable tag words the reason of their choice. These tag words or, alternatively, the direction or the distance between the previous guesses can be given to the other player as hint. A set of screenshots of the interface is reported in Fig. 3.2.

There is an interest in the results of the game also from a game theory point of view but our purpose is to get an annotated map of the focal points of the city. This map it is important because gives us information about urban dynamics, telling us about the most important aggregation points. This game will be also used to make polls about urban environmental condition. Through Blindate we can also ask people to meet in the most polluted place, in the most quiet place, or in the most safe place. In this way, we will actually make polls in the shape of a game about pollution or noise getting people to contribute better and more.

2012 © Copyright lies with the respective authors and their institutions

Figure 3.2: A set of screenshots of the Blindate interface.

Preliminary results

Until now Blindate gathered almost one thousand guess for the city of Rome. Even if the number of results is small we can start to figure out in what direction they will point at. First of all we can draw, and report in Fig. 3.3, a first map of Rome focal points. In this map we can see some obvious focal points such as famous squares, crossroads or bridges but there is also some surprise: the Coliseum is not a focal point, because it is too large, and actually it is quite rare to give a rendezvous at the Coliseum without further specifications. Going more deep into the dynamics of the game, we analyzed the ratio of winning matches at each turn, finding that the 55% of the players matched at the first guess. Then, of those who arrived at the second turn, only the 35% matched, a quite lower value. At the third turn the value slightly increases, at the fourth and at the fifth there is a very small value. The results are shown in left part of Fig. 3.4. This seems to point out the existence of a shared knowledge about the focal points, because people match very often at the first attempt. In the following attempts they seem to worsen their performances, which is apparently a paradox, since, after the first attempt, they begin to get information about the other player's guesses. To analyze better this aspect, we studied also how people improve their result during a match. The right part of Fig. 3.4 shows, for each of the five turns, the average distance between the guesses of the two players. At the first turn the distance is quite big but soon it decrease at the second turn

Figure 3.3: Rome map of focal points drew with the position guessed in Blindate. Redder points indicate an higher density of guess, corresponding to a focal point.

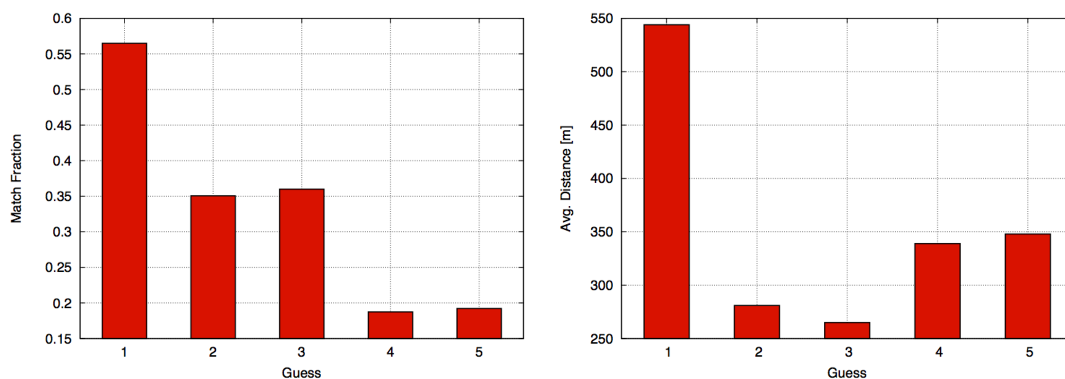


Figure 3.4: In the left graph, the ratio of winning matches at each turn (55% of match at the first turn; of the remaining 45% the 35% matched at the second turn and so on). In the right graph, the average distance between the guess of the two players at each turn of the game.

2012 © Copyright lies with the respective authors and their institutions

and reaches a minimum at the third turn. Then, at the fourth and fifth turn it increases significantly, meaning worse performances. Considering the fact that, after each guess, some information about the other player choice was given to the player, the worsening of the performance may point at the fact that not always the information its useful. More information may not correspond to better results. This qualitative result is somehow surprising and will be more deeply analyzed in the future, when more data will be available.

3.2.2 Joe's City Race

"Joe's City Race" is a web game designed to facilitate the analysis of the response of individuals to environmental information. For this, players are asked to draw a route between two points in a city, having local information available. At the moment, the game is implemented with traffic data, however, pollution data from the test cases will be added to the game. Two beta versions of the game are currently available at www.xtribe.eu, one single- and one two-player, developed using the ET platform. These are based in four cities, Turin, Rome, London and Dublin. Real traffic data has been obtained from the Dublin City Council and Octotelematics (for Turin), while for the other two cities data has been generated using open Street Maps and Google Maps.

Through the game responses to several questions are aimed for. One such question is how much information does a player need in order to observe a change in behaviour. In real life, how much information on the state of immediate neighbourhood is required for the citizen to be able to optimize the route. For this, in a single-player version, the platform displays different amounts of traffic information in each game, which will allow for an analysis in this direction to be performed. This is useful both from the social science point of view and for optimising future applications that offer visualisation of routing and traffic information.

Secondly, the behaviour in a context with social influence can be analysed. Specifically, in a multiplayer version, players can be aware or not of the movements of their opponents. If imitation appears (as it does in many real life situations), routes for players that could see one another will be very similar. However, the imitation behaviour could contradict the purpose of the game, which is minimising driving distance and time, so an analysis of the different strategies could give insight on the trade-off present in the player population.

Furthermore, a virtual traffic dataset is generated, based on the routes selected by users. This, analysed in comparison to the real data, can enable identification of traffic features related to street network topology. Also, the overall response to the real traffic displayed can be studied, showing whether avoidance of traffic can create jams in other locations of the city.

Future versions of the game will also show pollution data, in order to understand the strategy, if any, used by people to minimise exposure to pollution while driving or walking.

The game

The single-player version of the game aims to analyse the effect of traffic information, and the extent of it, on the routes players choose. For this, the game consists of two stages. The game starts with the user selecting a city to play in (Figure 3.2.2). This leads to the first stage of the game, when two points are given on a standard Google map, and users are asked to draw their preferred route. Users draw a route by selecting successive points on the map, in an active green area (Figure 3.2.2), and then click on the destination to finish. At this point, the second stage starts, when the same task is repeated, but with traffic information also displayed, colour coded for each street (red - high to green - light traffic) (Figure 3.2.2). The user has to select again a route, and the change in strategy will be the effect of traffic information. Different amounts of such information are displayed (i.e. from a small to a large area around the current location of the player), in order to enable the study of how player behaviour and performance is influenced by the

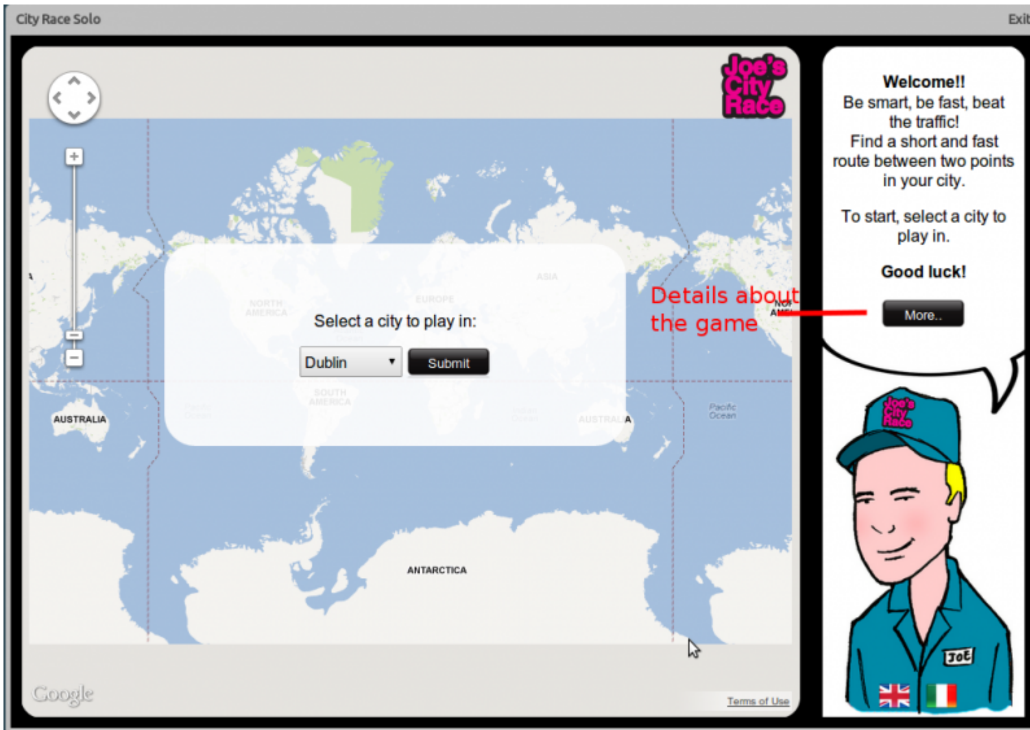


Figure 3.5: Joe's City Race: select location.

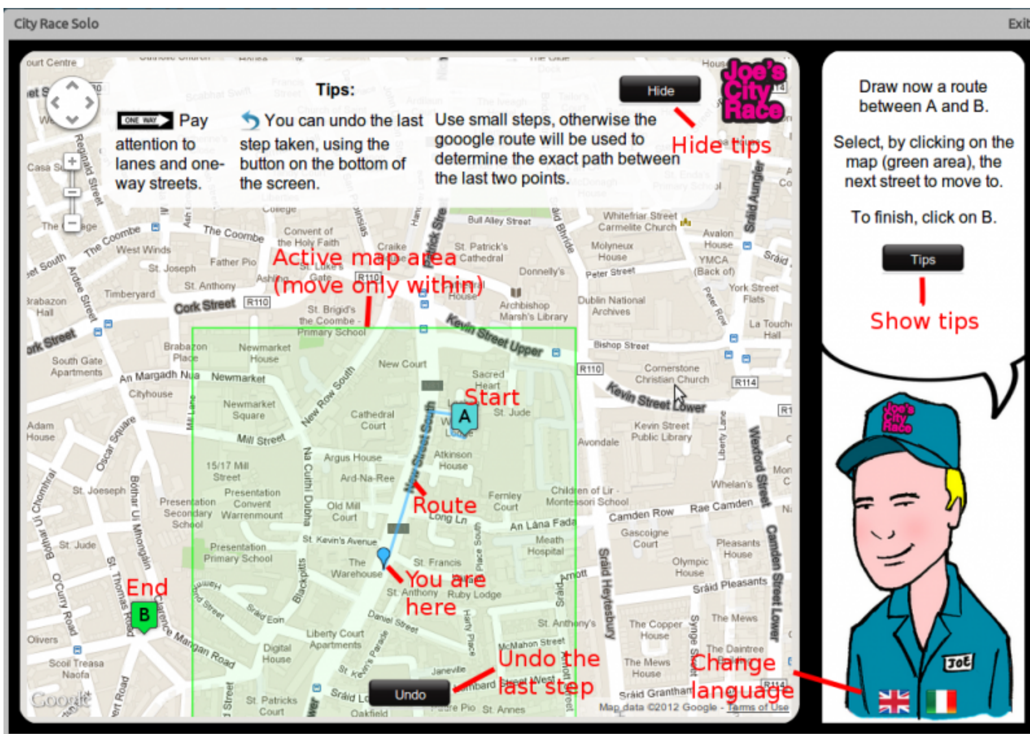


Figure 3.6: Joe's City Race: single-player, phase I, no traffic information.

2012 © Copyright lies with the respective authors and their institutions

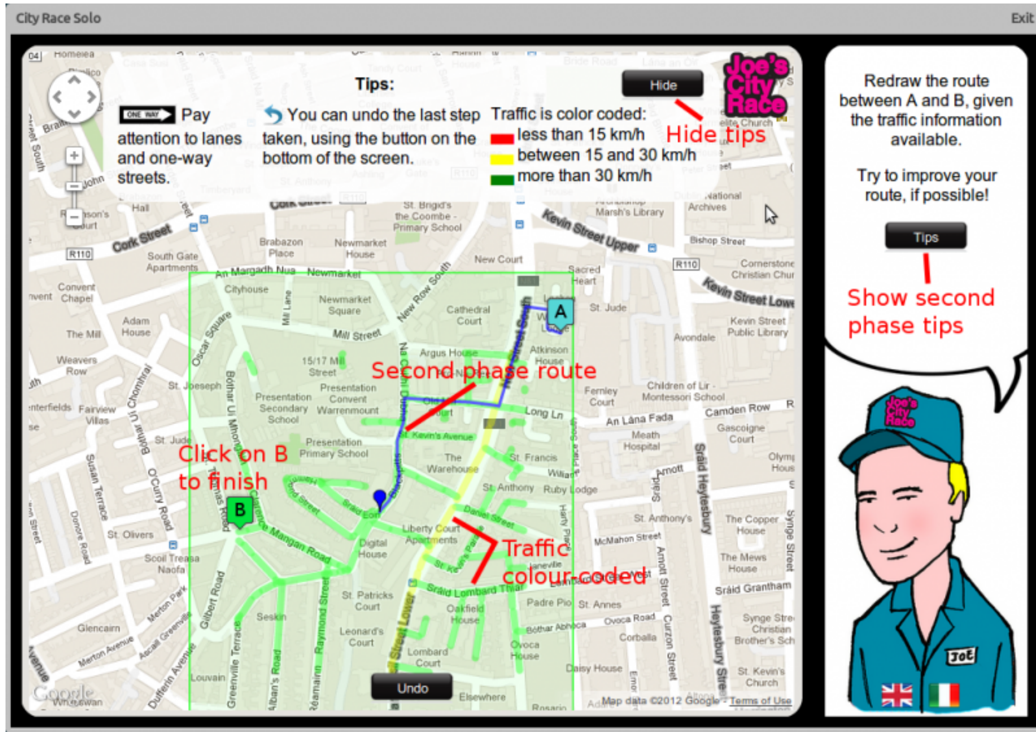


Figure 3.7: Joe's City Race: single-player, phase II, traffic displayed as colours on the streets.

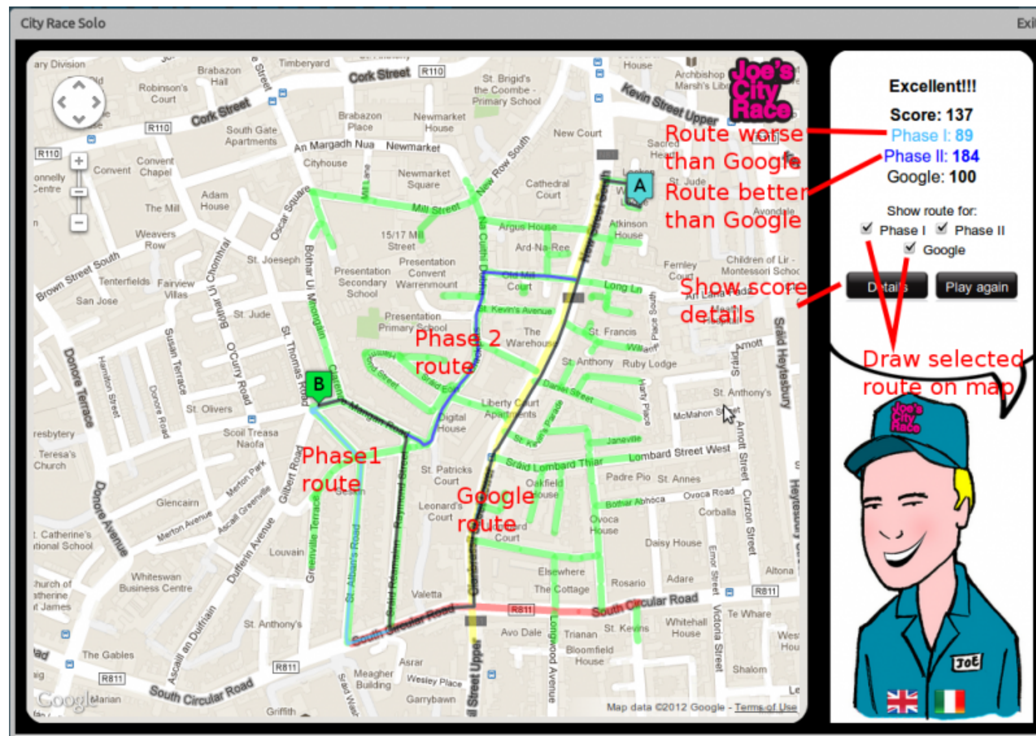


Figure 3.8: Joe's City Race: score for single-player game.

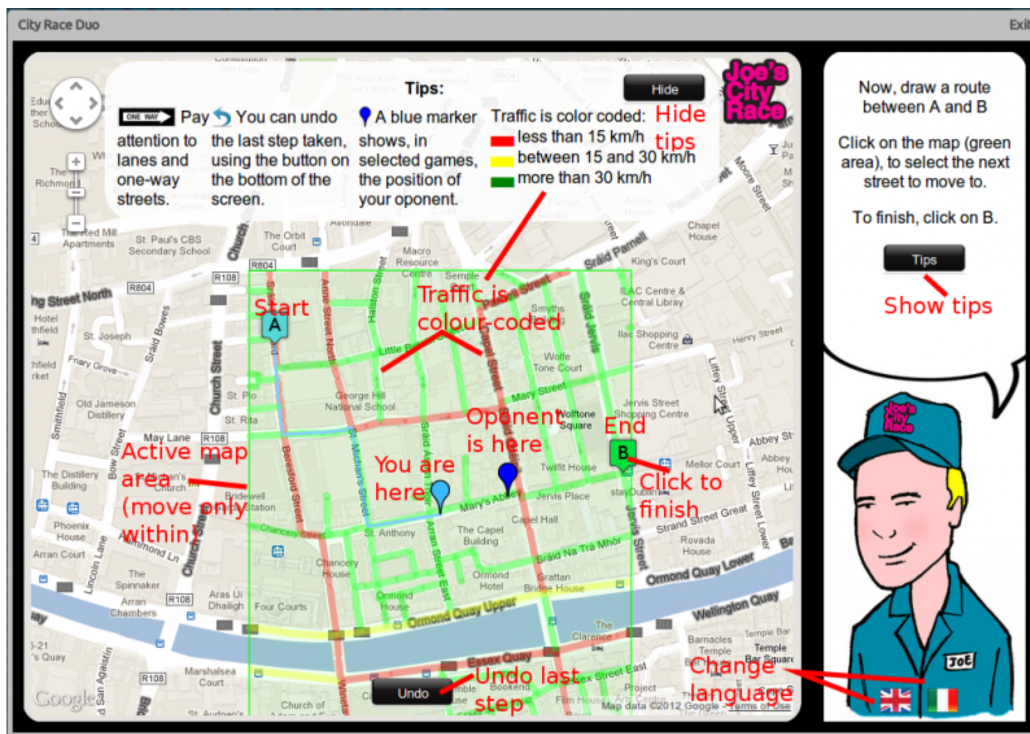


Figure 3.9: Joe's City Race: multi-player.

amount of information. When the second route is complete, a score is given to the user, which compares the distance and time they obtained to that of the Google Directions Service. A score larger than 100 indicates a route better than Google, while lower than 100 signifies the route is not superior. Also, the three routes (stage 1, stage 2 and Google) are displayed for comparison (Figure 3.2.2).

The multi-player version of the game aims to analyse social influence in choosing a driving route. The game is organised as a race between two or more players, where the best route wins. In (randomly) selected games, players can see the movements of their opponents (Figure 3.9), and choose to follow them, or, on the contrary, select a very different route to the destination. The behaviour of players can thus be analysed and the social effect studied. The game is played in a similar fashion as the single-player version, where each player chooses and submits a route between two points. The score is then calculated for each player, and the highest wins. Again, the score screen shows the three routes for comparison (player 1, player 2 and Google - Figure 3.10)

The game implementation consists of a few components, which communicate through the Experimental Tribe platform. These are the manager, database and client. The manager extracts traffic information from the database and sends it to the client, to be displayed on the map. The client handles all user input and makes requests to the manager accordingly. Due to large amount of traffic data, several levels of optimisation had to be implemented in order to facilitate data display in a timely manner. These include database query and index optimisation and server side data processing.

Preliminary results

A first analysis concentrates on the behaviour shift when players have traffic information available. Figure 3.11 shows the fraction of single-player games where the route selected by the user has improved, disimproved or remained the same from the first game stage (without traffic) to the second (with traffic). This is displayed for different amounts of information available on the map. A

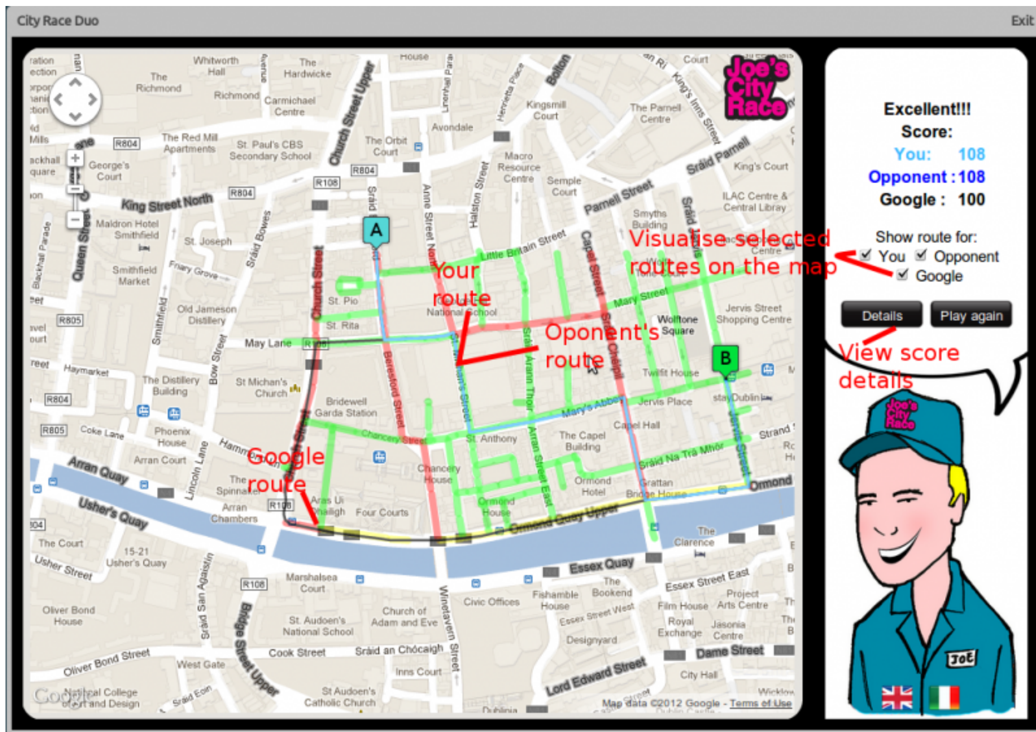


Figure 3.10: Joe's City Race: score for multi-player game.

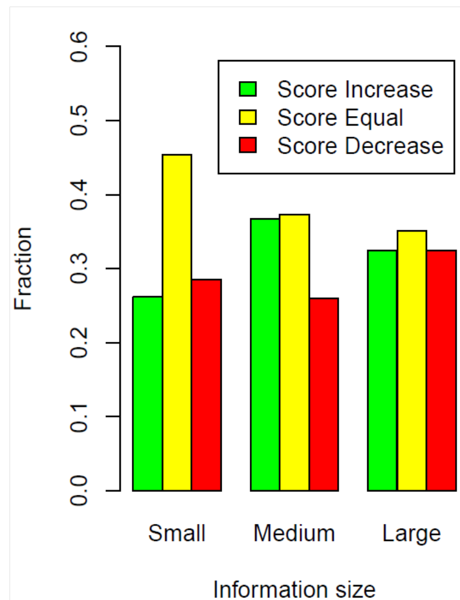


Figure 3.11: Effect of the amount of traffic information on player performance.

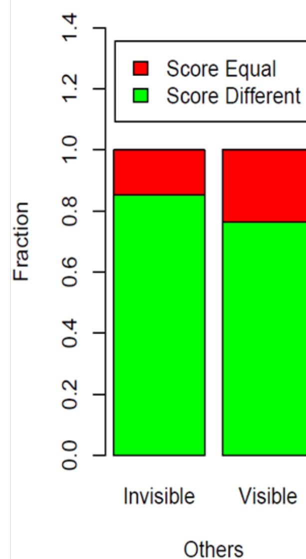


Figure 3.12: Imitation during multi-player games.

first indication is that the number of players changing their routes when traffic information becomes available increases with the amount of information. This shows that traffic has to be shown for a larger area in order to observe a larger behaviour shift. Secondly, the size of the traffic information displayed affects the ability of users to improve their route. Information which is too restricted leads to the least improvement, probably due to “greedy" choices. On the other hand, information which is too large does not provide the best improvement, with medium information being the best. In the case here, medium information means an area as large as the one where the user can move. This indicates that there is a limit to how useful enlarging the amount of information is, and this is maximised through moderate amounts of information.

Secondly, we are interested in the amount of imitation during two-player games. This gives indication on the influence of society in enhancing environmental awareness. Figure 3.12 shows the percentage of games where the two players choose the same route, both when they can see each other on the map, and when they can not. This shows an increase in similarity between routes when players are aware of the other's position, indicating that some imitation does appear. Additionally, we have analysed the distribution of scores with and without social influence (Figure 3.13). Although these are preliminary results, indications exist that the score are improved, overall, when players can see each other , showing that imitation and the social effect is useful.

3.3 Technical details

ET has been designed with a modular structure through which most of the complexity associated to running an experiment is hidden into the ET Server, while the experimentalist is left with the only duty of devising a suitable interface for the actual experiment. In this way most of the coding difficulties related to the realization of a dynamic web application are already taken care by the ET Server and the realization of an experiment should be as easy as constructing a dynamic web page.

2012 © Copyright lies with the respective authors and their institutions

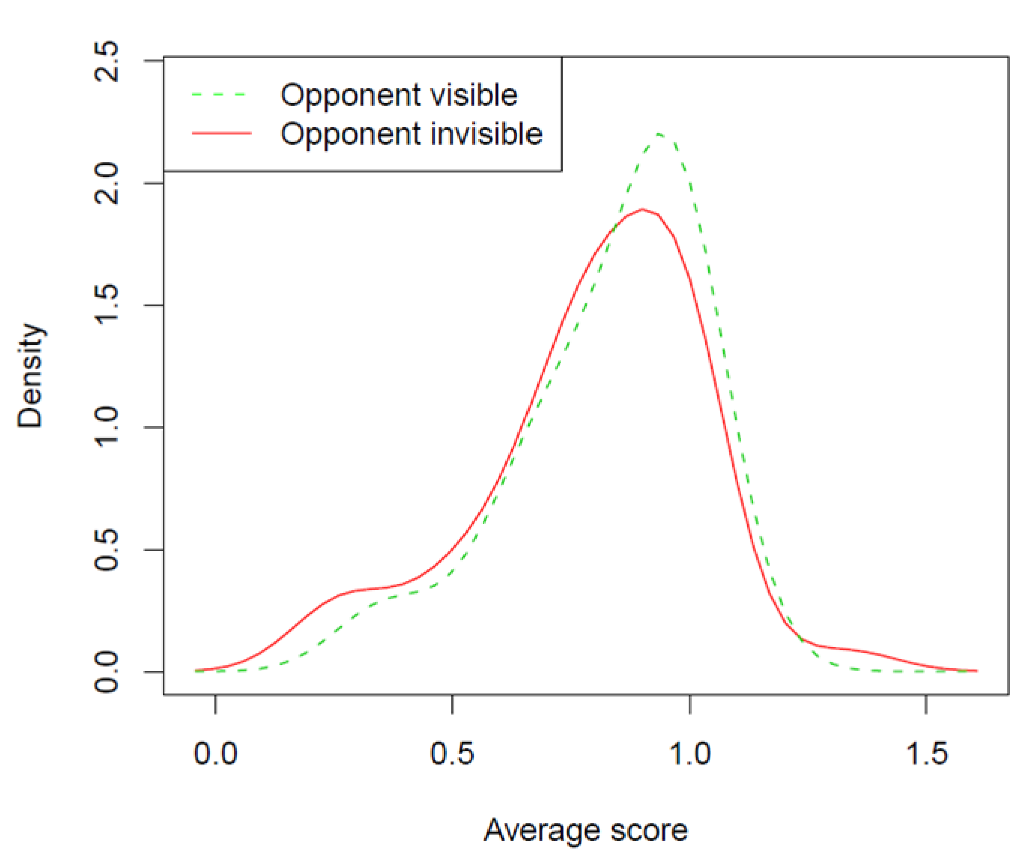


Figure 3.13: Score for multi-player games.

3.3.1 Basic entities

The ET web site is build upon Drupal [3] and, beside system administrators, there are two different kinds of users on it:

1. the experimentalists who publish individual experiments through ET;
2. the players who participate in one or more individual games.

Each published game is as a combination of two intercommunicating parts: the User Interface (UI) and the Game Manager (GM). The UI is what is visible to players: it is a web page that displays the game state and receive user input. The GM is in charge to implement the game-dependant centralized logic, such as choose initial data for each match, gather and process user activity, assign scores, etc. The communication between the UI and the GM is mediated by the ET Sever.

3.3.2 Technologies for development and communication

The UI is hosted on the ET Server and can be implemented using any modern web technology such as HTML, JavaScript, CSS or Flash. It is given access to a set of JavaScript API that allow the experimenter to easily implement asynchronous bidirectional communication with the ET Server and, in turn, with the GM. The actual communication is internally carried out by means of WebSockets; if the user browser does not support them, several fallback techniques (such as http long polling or multipart streaming) are used. In order to keep this short-message exchanging efficient we used NodeJS [4] for the server side implementation in conjunction with the Socket.io

[3] http://drupal.org/
[4] http://nodejs.org/

JavaScript library [5]. All this complexity, as well as all browser-dependant choices, are hidden to the experimenter who can send messages by calling a single JavaScript function and can receive asynchronous messages by registering a callback function.

The GM will be hosted by the experimenter on his own server. This way the scientist is given full control over the experiment and can directly collect and access resulting data in real time. The GM can be developed with any server side technology such as PHP, ASP, Perl, Java, NodeJS, etc. The communication with the ET Server takes place through the HTTP protocol and all exchanged messages are coded as JSON strings [6] received as post variables and sent back in the body of the response. If required the GM can initiate a message exchange phase by contacting a dedicated URL of ET Server over HTTP.

Besides a restricted set of system messages, the game internal protocol is fully elaborated by the researcher to exchange game-dependant messages between the GM and the UI. Each message is characterized by a topic (a string) and can carry any kind of parameters, ranging from single values to arbitrarily complex arrays and objects.

Remarkably, the ET Server automatically takes care of the game setup phase: it allows users to join each specific experiment and, when enough players are ready, the server creates an instance of the experiment and notify the game start to the GM and all the players, leading them to the actual game UI. The server also manage all message exchanging based on instances in order to allow instance-wide message broadcasting functionalities.

The platform will also handle errors and exceptions: e.g., if a player disconnect unexpectedly, the system will detect it and notify the instance abortion to the remaining players and to the GM. Since there is no direct communication between the GM and the UI, the GM will experience no trouble at all.

A schema of the communication between the ET Server, the player and the GM is shown in Figure 3.14. Communication between the ET Server, the player browser (left side), and the GM (right side). ET Server takes of the join phase automatically then it mediates the communication between the UI and the GM, while handling errors and exceptions

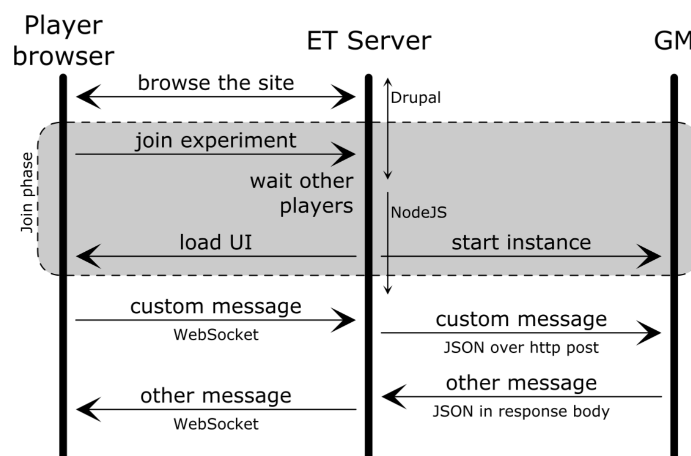


Figure 3.14: Communication between the ET Server and the GM.

3.3.3 Game setup

For each game the ET platform host a page containing game rules, images and a description of the scientific objectives related with the experiment, possibly reporting obtained results. From this

[5] http://socket.io/
[6] http://www.json.org/

2012 © Copyright lies with the respective authors and their institutions

page a player can learn more about the experiment and decide to participate by playing it.

Since the game has been created for research purposes, the researcher is interested in all sort of statistics related to players. Therefore anonymous player information can be made available to the UI and the GM.

While publishing a game, the researcher have to declare how many player are required to start an instance of the experiment. He may also be interested in grouping and filtering players for specific purposes, e.g. according to their age, gender, geographical location, nationality, etc. To this end, ET handles a user registry in which players will be allowed to register, if required, and play while the system would maintain all the information about them such as scores, ranks, etc. Furthermore, based on this information, the system may also grant the access to the game only to certain profiles.

Being in charge of the handling of the user registry, the system would also spare the researcher from dealing with privacy and security issues since all data will be properly anonymized and, possibly, encrypted. However, by default, it is still possible for unregistered users to access the games. Filters are applied only if set by the researcher.

3.4 Further developments of the platform

To summarize, ET handles all the aspects of the realization of web experiments that does not concern directly the game itself, thus allowing the researcher to focus only on the core of the experiment, leaving the rest to the system.

The ultimate aim of the project is to allow researchers working in different fields, who lack computer science expertise, to create web-based experiments and games. In order to achieve this goal, the first step is to create a set of “default” GMs for games corresponding to the most standard types of web experiment, such as surveys or coordination games. At the moment, the only “default” GM available in the platform simply broadcasts to all the players the message received from each one. The following step will be the realization of a set of graphical tools that will make it possible to set up a web experiment without writing a single line of code, e.g. a drag and drop-like system that allows the development of interfaces and the creation of the relative logics.

In the long term, the platform will also come in help when dealing with another typical issue of web experiments: the recruiting. It is often quite difficult to gather a critical mass of “suitable” players, but since the game is hosted on the platform, and will be shown on its main page, other players already involved in other games would probably join. We expect a community of players to gather on the platform playing different games and also giving researchers feedback about their experiments. We also expect researchers to spontaneously aggregate into communities, sharing advices and best experimental practices with each other.

Bibliography

S Arnstein. A ladder of citizen participation. *JAIP*, 35(4):216–224, 1969. URL http://lithgow-schmidt.dk/sherry-arnstein/ladder-of-citizen-participation.html.

LB Chilton, CT Sims, M Goldman, G Little, and RC Miller. Seaweed: a web application for designing economic games. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '09, pages 34–35, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-672-4. doi: http://doi.acm.org/10.1145/1600150.1600162. URL http://doi.acm.org/10.1145/1600150.1600162.

S Cooper, F Khatib, A Treuille, J Barbero, J Lee, M Beenen, A Leaver-Fay, D Baker, Z Popović, and Foldit Players. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307): 756–760, August 2010. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature09304.

V Crawford, U Gneezy, and Y Rottenstreich. The power of focal points is limited: Even minute payoff asymmetry may yield large coordination failures. Am.Ec.Rev., 98:1443–1458, 2007.

D Fischer and et al. Planet Hunters: The First Two Planet Candidates Identified by the Public using the Kepler Public Archive Data. *arXiv:1109.4621v3*, 2011.

MF Goodchild. Citizens as Voluntary Sensors: Spatial Data Infrastructure in the World of Web 2.0. *International Journal of Spatial Data Infrastructures Research*, 2:24–32, 2007.

F Khatib and et. al. Crystal structure of a monomeric retroviral protease solved by protein folding game players. *Nat Struct Mol Biol*, 18:1175–1177, 2011.

Winter M and Duncan JW. Financial incentives and the "performance of crowds". KDD-HCOMP '09, Paris, France, June 28 2009.

J Mehta, C Starmer, and R Sugden. The nature of salience: An experimental investigation of pure coordination games. *Am.Ec.Rev.*, (74):658–673, 1994.

BA Nosek, MR Banaji, and AG Greenwald. E-research: Ethics, security, design, and control in psychological research on the internet. *Journal of Social Issues*, 58:161, 2002. doi: 10.1111/1540-4560.00254.

G Paolacci, J Chandler, and P Ipeirotis. Running Experiments on Amazon Mechanical Turk. *Judgment and Decision Making*, 5(5):411–419, 2010.

E Paulos, RJ Honicky, and B Hooker. Citizen science - enabling participatory urbanism. In M. Foth, editor, *Handbook of Research on Urban Informatics: The Practice and Promise of the Real-Time City*, pages 414–433. IGI Global, 2009. URL http://www.urban-atmospheres.net/CitizenScience/.

MJ Salganik and DJ Watts. Web-Based Experiments for the Study of Collective Social Dynamics in Cultural Markets. *Topics in Cognitive Science*, 1(3):439–468, 2009. ISSN 1756-8765. doi: 10.1111/j.1756-8765.2009.01030.x. URL http://dx.doi.org/10.1111/j.1756-8765.2009.01030.x.

2012 © Copyright lies with the respective authors and their institutions

T Schelling. *The strategy of conflict*. Harvard UP, Cambridge, Mass., 1960.

S Siddharth and JW Duncan. Cooperation and contagion in web-based, networked public goods experiments. *PLoS ONE*, 6(3):e16836, 2011.

L von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006. ISSN 0018-9162. doi: http://doi.ieeecomputersociety.org/10.1109/MC.2006.196.

L von Ahn and L Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM. ISBN 1-58113-702-8. doi: http://doi.acm.org/10.1145/985692.985733.