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)

D2.2: Final version and report on the web-based infrastructure

---

Period covered: from 01/09/2012 to 28/02/2014
Start date of project: March 1st, 2011
Due date of deliverable: Apr 30th, 2014
Distribution: Public

Date of preparation: 28/02/2014
Duration: 36 months
Actual submission date: Apr 30th, 2014
Status: Draft

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

Executive Summary

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 correctness and performance of the system, as well as easy management and reuse.

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. The *EveryAware* platform has been explicitly designed to support subjective impressions in conjunction with sensor data acquisition by introducing an extendable data concept. A central server efficiently collects, analyzes and visualizes data sent from the arbitrary sources. The platform offers a highly flexible way to store and exchange data for Internet of Things applications. A wide variety of meta, location, and content information which can be attached to any data point, a flexible *data processor component* as well as an efficient storage structure are the keys to this task. This mechanism provides the unique ability to enrich data with contextual information explicitly including subjective impressions.

Additional to this data collection infrastructure, a gaming platform called *Experimental Tribe* (XTribe) has been developed focusing on collecting even more subjective data. XTribe is 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.

Outline of the document

After a brief overview of the subject in Chapter 1, in Chapter 2, we start by describing how the *EveryAware* system handles the combination of objective and subjective data on a conceptual level by introducing several key data modeling concepts and continue with an overview of the storage and processing framework on a technical level. In Chapter 3 we introduce the REST API which is responsible for setting up interface to and from external services. In Chapter 4 we give an overview of the web interface including a connection to social platforms like Twitter[1] or Facebook[2] and several global as well as personalized interactive visualizations. The final Chapter 5 focusses on the X-Tribe platform for gaming, a well-established way to engage humans in experiments.

[1] `https://twitter.com`
[2] `https://facebook.com`

Dissemination of the Results

This Deliverable 2.2 can be considered as the continuation of Deliverable D2.1 where the first version of the *EveryAware* platform has been introduced. The platform was extended continuously, especially in order to support the large amount of visualization demands imposed by the case studies and demos. Two main applications where hosted by the *EveryAware* platform during the project, namely *AirProbe* and *WideNoise Plus*. Their capabilities have been presented to the scientific community in several articles [Atzmueller et al., 2012, 2014; Becker et al., 2013b]. Other applications are currently being developed.

The platforms are online accessible at `http://airprobe.eu/` and `http://widenoise.eu/`, resp. The smartphone apps are available in Google Play[3][4] and Apple Store[5].

The data from the *WideNoise Plus* application were thoroughly analysed and published in several articles [Atzmueller and Mueller, 2013; Becker et al., 2013a], while the study carried on the *AirProbe* data will be submitted soon to an international peer-reviewed journal (PLOS ONE).

The case studies supported by the *EveryAware* platform include but are not limited to several community projects around Heathrow Airport in the *WideNoise Plus* context and the APIC International Challenge as well as a school project in the *AirProbe* context. The APIC International Challenge is online accessible at `http://www.everyaware.eu/APIC/`.

The details about the XTribe platform hosting the web-based experiment part of the AirProbe International Challenge were presented to an international conference and published as proceeding [Caminiti et al., 2013]. The XTribe platform is online accessible at `http://xtribe.eu/`.

[3] `https://play.google.com/store/apps/details?id=eu.everyaware.widenoise.android`
[4] `https://play.google.com/store/apps/details?id=org.csp.everyaware`
[5] `https://itunes.apple.com/app/id657693514`

Contents

List of Figures

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.

The *EveryAware* platform has been explicitly designed to support subjective impressions in conjunction with sensor data acquisition by introducing an extendable data concept. A central server efficiently collects, analyzes and visualizes data sent from the arbitrary sources. The platform offers a highly flexible way to store and exchange data for Internet of Things applications. A wide variety of meta, location, and content information which can be attached to any data point, a flexible *data processor component* as well as an efficient storage structure are the keys to this task. This mechanism provides the unique ability to enrich data with contextual information explicitly including subjective impressions. Different collection concepts like sessions to represent time-interval-based entities and feeds to organize data points in a continuous way allow to further introduce semantic relations. This enables the web interface to provide different semantically enriched views on the data aggregating data globally as well as on a personal level. Additional to this data collection

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

infrastructure, a gaming platform, called XTribe has been developed focusing on collection even more subjective data.

In Chapter 2, we start by describing how the *EveryAware* system handles the combination of objective and subjective data on a conceptual level by introducing several key data modeling concepts. We continue with describing the technical details necessary to implement a highly efficient and distributed system including database setup as well as data processor components. The resulting system can be seen as the primary data backend for most of the applications developed within the EveryAware project.

In Chapter 3 we introduce the REST API which is responsible for setting up interface to and from external services. Smartphone applications as well as the web frontend use these interfaces to comminute with the platform in order to store and retrieve data as well as semantically enriched aggregations and statistics.

In Chapter 4 we give an overview of the web interface including several global as well as personalized interactive visualizations. We provide semantically enriched data to the users by utilizing a data collection process which emphasizes the use of social interaction. This includes a connection to social platforms like Twitter[2] or Facebook[3].

The final Chapter 5 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, the corresponding API and web interface together with the gaming platform are the 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.

[2] `https://twitter.com`
[3] `https://facebook.com`

Chapter 2

Architecture

The platform has been designed to facilitate the combination of sensing technologies, networking applications and data processing tools. This enables users to collect and visualize environmental information and at the same time augment the collected data with arbitrary information explicitly supporting subjective context. Additionally the data processing engine allows for the application of dedicated data mining and knowledge discovery algorithms in order to fully exploit the synergies of a central data storage and the wide variety of objective and subjective information. Our platform is based on the framework introduced in [Becker et al., 2013b] and extends it to provide the functionality needed for our case studies.

The platform comprises two main layers. The conceptual layer defines the basic entities and features the *EveryAware* system supports. It introduces an innovative framework to collect, process and retrieve data and features straight forward usability, flexible access control as well as a powerful data extension mechanism. The implementation layer introduces the data storage layout as well as the data processor. Both ensure high availability as well as generic data handling and processing and enable advanced processing methods inferring missing data or learning approaches for extracting information from the raw data resulting in higher level semantic information. The conceptual and the implementation layer are described in the following.

2.1 Conceptual Layer

The conceptual layer defines the basic entities and features the *EveryAware* system supports. The core concepts are data points with descriptions, sessions and feeds. Data points and sessions can be extended by other data points. Single data points can be claimed by users, i.e., the user can claim ownership. Feeds are access restricted and can define visibility levels. See Figure 2.1 for an overview.

The basic entities of the *EveryAware* system are data points. Each data point consists of a set of fixed description attributes in addition to the actual data. These attributes ensure the processability as well as dynamic querying of arbitrary content including binary data like photos or videos. The description attributes are divided into three categories:

1. Meta attributes are attributes which allow to keep track of data independent information like received time, recording time, device ID, session ID, etc.

2. Geo attributes make it possible to record the location of the sample being taken including longitude and latitude as well as accuracy and the provider of the location fix.

3. Content attributes describe the content and its format. They help the system to further process the data. These attributes include the data type (e.g., air, noise, image) and format (e.g., JSON, XML, PNG).

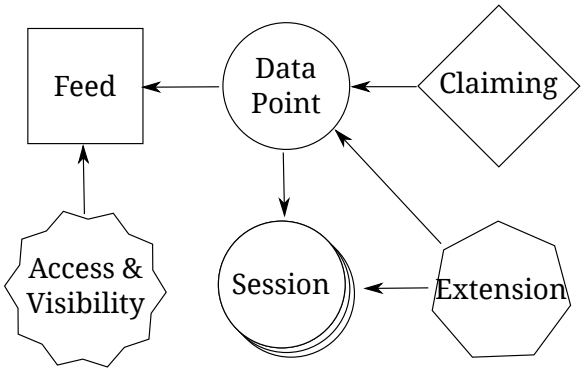


Figure 2.1: Overview of the core concepts of EveryAware architecture.

Sessions are collections of data points limited to a fixed timespan. Sessions allow to introduce semantic entities such as “my way to work” or “a stroll in the park”.

Data points as well as sessions can be extended with additional information using other data points. This makes the data representation very flexible and inherently support the augmentation of objective data with a semantic context. One application is tagging. Sessions and data points can be tagged by extending them using tag data points referring to the respective data point or session IDs to be tagged. Tagging is not only restricted to actual text-tags but can be any kind of data including videos, sound files or air quality measurement. Using this scheme, it is also possible to update data points as well as sessions after they have been sent without losing the original data. Since no raw data is deleted, this also allows to always access the version history of a data point.

Data points can be organized in feeds. A data point is always part of the global feed, but can also be pushed into several other feeds. Users can contribute to existing feeds or create their own feeds. While useful for organizing data points, feeds also allow to attach data points to real world entities such as major events like music festivals, places like the Eiffel Tower or portable things like a smartphone. Feeds can be access restricted and a visibility level can be specified for each data point in a feed.

Feeds can be *open* or *closed* concerning read and write access, where *write access* refers to the possibility of adding new data points to a feed. Open feeds are accessible by everyone including anonymous users. Closed feeds are only accessible by a limited set of users (*members*). The global feed is always open for read and write access. It contains all data points without exception. The access restriction allows users to create feeds and share them with friends or other interested users without making their data publicly available.

Since privacy is an issue and users may want to contribute in different ways to the collected data and corresponding statistics which might be derived from it, the *EveryAware* system introduces visibility levels for each data point in a feed. This is particularly important since all data points are part of the global feed. There are four visibility levels:

1. *details* allows everyone who has access to the feed to see the raw content as well as the description attributes of the data point.

2. *statistics* restricts the data point to be considered in user statistics derived from the data points in the feed, e.g., average values for the corresponding user.

3. *anonymous* restricts the data point to only be considered in overall statistics derived from the data points in the feed, e.g., average values for an area or timespan. No association with the user is possible.

4. *none* allows only the owner of the data point to access the data point and its description attributes.

Claiming is a concept which allows anonymous contribution to the *EveryAware* system while giving the user possibility to claim data points as soon as she decides to register a user account. This makes it convenient to contribute data to the *EveryAware* system and provides some level of anonymity since no previous registration process is required. At the same time the collected data is not lost for the user but can be accessed at a later time. Claiming works by exploiting the device ID, which is usually sent as part of a store request. If a registered user sends a data point with a device ID, she can claim all data points she has sent before with the same device ID.

2.2 Implementation Layer

The implementation layer realizes the the conceptual layer based on an advanced storage and application structure. This structure consists of several components:

- the storage itself,
- web application components for handling data endpoints receive and retrieve data, as well as a
- data processor component which processes and semantically enhances incoming data.

Currently the storage layer is based on MySQL[1]. MySQL supports geo-data enabling efficient storage and retrieval of spatial data points. Storage is organized as a pipeline enabling a distributed manner of storing, processing and accessing the data. A more detailed description of the pipeline is given in the next sections.

The web application is implemented within the Servlet container Apache Tomcat and is based on the Spring MVC Framework[2]. Thus, the underlying storage structure is easily interchangeable by utilizing the Inversion of Control paradigm. Furthermore the Spring Social module allows to easily and modularly add connectors to many a set of social services such as Facebook, Twitter or LinkedIn. Currently, the web application focuses on Twitter and Facebook allowing to share different content of the web page.

The web application is also responsible for providing several REST endpoints as described in Chapter 3. Those REST endpoints allow to upload data to the EveryAware platform (for example using the WideNoise or the AirProbe application) and provide several possibilities to retrieve the uploaded data itself as well as several statistics, summaries and higher level semantic information (see Chapter 3 and Chapter 4). At the same time a more raw yet also a more detailed version of the data is provided to the Consortium via an SQL dump.

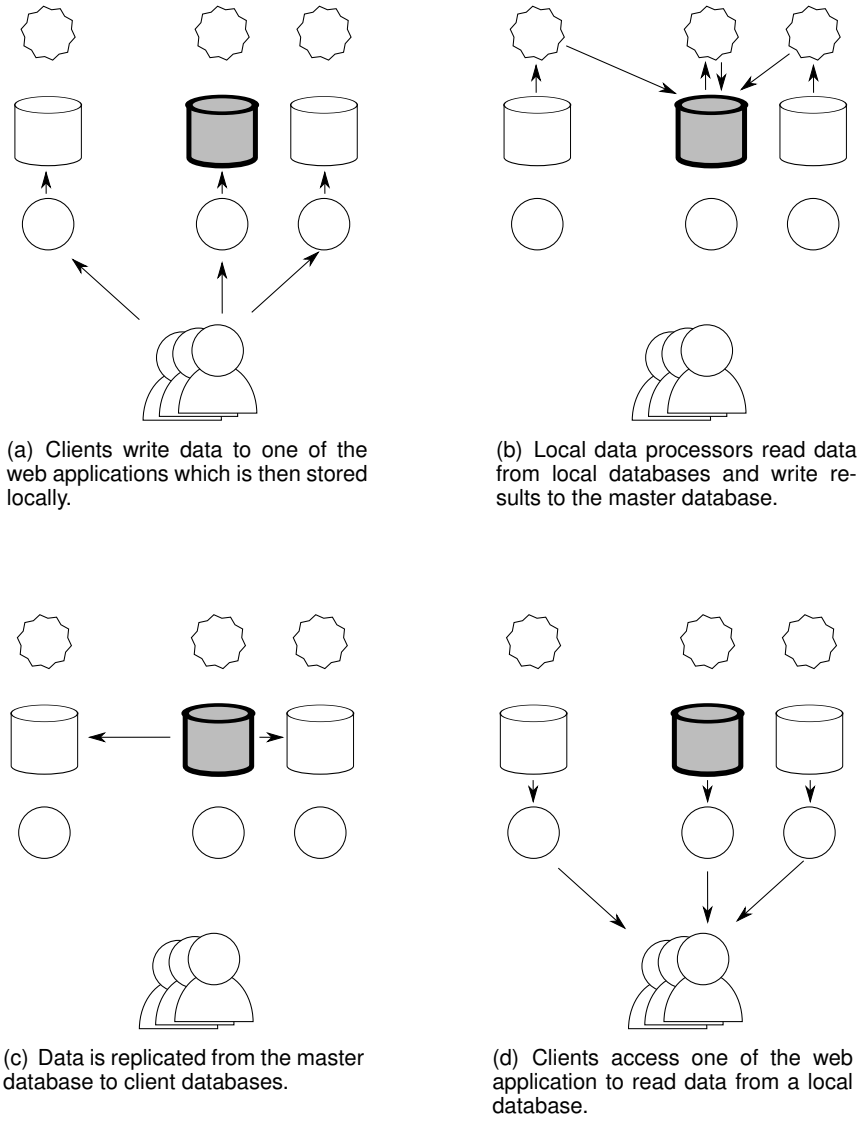
The data processor has several components. The main component is responsible for cleaning, processing and enhancing incoming data in real time. Thus this is considered the real time component. Building upon the data provided by real time component are the batch components. The batch components are responsible further higher level semantic information such as grids and clusters or heatmaps which are more time consuming to calculate.

In the following sections, we will elaborate on the data pipeline used to efficiently store and process data in a distributed manner and then go into more detail about the data processor.

2.2.1 Data Pipeline

The data pipeline of the EveryAware platform is designed to store large amounts of data and provide highly available, responsive storage and query capabilities. Highly available, responsive end-

[1] http://www.mysql.org
[2] http://spring.io



(a) Clients write data to one of the web applications which is then stored locally.

(b) Local data processors read data from local databases and write results to the master database.

(c) Data is replicated from the master database to client databases.

(d) Clients access one of the web application to read data from a local database.

Figure 2.2: General data flow of the EveryAware architecture. Columns represent logical nodes (i.e. may be located on one or several machines). Circles are web servers, cylinders are databases and stars are data processors. The master database has a grey background.

points minimize data loss and enable our web application to provide a real-time user experience. The general structure of the pipeline is visualized in Figure 2.2.

The pipeline is divided into several logical nodes. There is one master node and several slave nodes. Each node consists of a web server, a local database and a data processor. The web server hosts different components of the EveryAware infrastructure. There are three components: write endpoints, read endpoints and a web application providing a user interface. In general, clients can read and write to any node (except if certain nodes are dedicated to certain tasks, like only serving REST endpoints for reading but not for writing), see Figures 2.2(d) and 2.2(a). On slave nodes the local database is a replication of the master node's database used for read access only. There is also a local data cache for data which is written to the slave node. The local cache is then read by the local data processor, processed and the saved to the master database, see Figure 2.2(b). The data processor component is described in detail in Section 2.2.2. Finally, the replication mechanism will distribute the new data to the slave nodes' databases 2.2(c).

This method allows to distribute workload between several machines, thus, enabling a highly avail-

able, responsive system. Note, that the described nodes are logical nodes. Workload can further be distributed by splitting logical nodes into a web application, a database and a data processor component. The web application can further be split into several components as mentioned above. Also, the database itself is not restricted to a database hosted on a single node but may be distributed itself enabling further workload distribution.

2.2.2 Data Processor

The *data processor* is responsible for parsing the received data, resolving extensions, apply knowledge discover processing steps, and augmenting them with additional semantic information from various sources.

Overview

The most important responsibility of our data collection platform is to reliably store the received data points. To ensure this, the storing procedure itself keeps computational overhead such as syntactic checks and other processing at a minimum. The received data is directly written to the input table. Only afterwards will the *data processor* start parsing, verifying and enhancing the stored data using a modularized approach. This gives us the following advantages:

- *High performance and availability*: Any computation in the endpoint would slow down the data reception. This is particular true for applications with large content and high transmission frequencies.
- *Flexibility*: We can accept literally any data, since the endpoint does not restrict the type of data sent via the REST API. Different data types are handled by a dynamically manageable set of *processing modules*.
- *Robustness*: Storing and keeping the data in its raw form enables us to recreate the processed content at any point in time.

In the second step, the *data processor* post-processes the data. Figure 2.3 shows the general architecture. The *data processor* reads data from the input table and combines it with information about ownership. Data is then parsed, interpreted and augmented using a chain of *processing modules*. This includes applying knowledge discovery steps in order to gain more insights about the data. The resulting information is stored in the so called content tables which are query-able from data access endpoints.

A special content table is the output table where all incoming data is stored in its raw format without applying advanced processing steps. The stored data includes any description attributes like location or visibility information. This allows for accessing any sent data even if not data processor module exists which is able to parse a specific data type.

This division of input and access storage enables support for replication of the input and output endpoints of the REST API. At the same time distributed processing of large amounts of incoming data is possible. In order to keep processing synchronized, the *data processor* sets a processing state for each processed data point based on the processing result (e.g., success or error).

Processing Procedure

The *data processor* constantly checks for data which needs processing: new data and data that should be updated. After retrieving the data to process, the *data processor* runs through its different components: the *module selector*, the selected *processing module* and the *storage handler*.

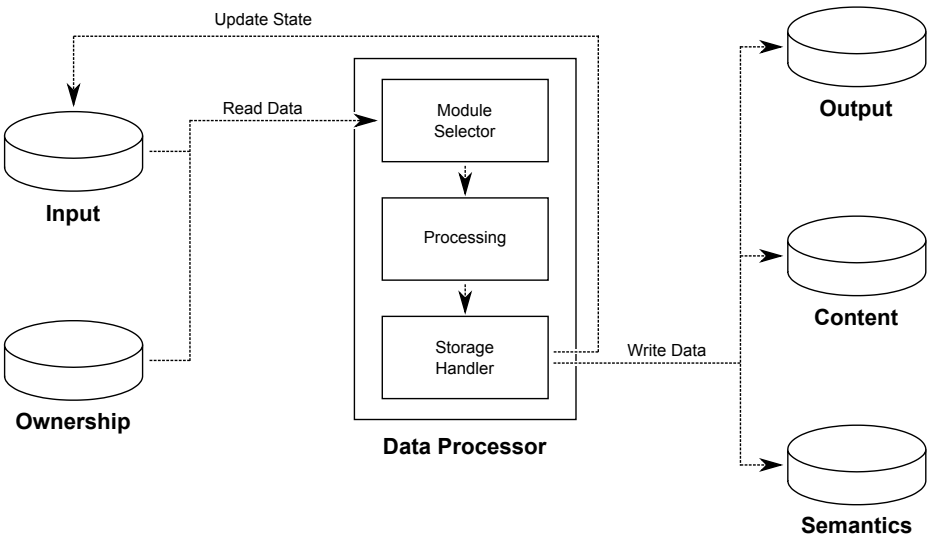


Figure 2.3: Architecture of the *data processor*.

The *module selector* selects a *processing module* from a priority-chain. The matching module is selected based on its data type defined by the descriptions attributes or by deriving the data type from the content.

The second component is a selected *processing module*. It extracts the actual data from the raw content (e.g., JSON) and possibly augments the data with additional information, calculates statistics, or handles missing information.

Then, the *storage handler* stores the resulting data in dedicated content tables, the output table, and possibly semantics tables. After the storage engine has finished writing the data to the corresponding tables, it adjusts the processing state of the data in the input table.

This architecture has several advantages. The priority-chain-approach in the *module selector* allows for flexibility in extending the *data processor* with additional *processing modules* on demand. The modularized approach in general makes it easy to deploy updates without risking to break the existing data.

The processing state set by *storage handler* for each processed data point is the key to flexible module extensions and ensures robustness against processing failures: Assuming a *processing module* is updated, the processing state of the corresponding data is reset. The *data processor* will then re-process only those marked data.

Processing Modules

There are several modules implemented for processing data. In this section we will highlight the *AirProbe* processing module which processes the data coming from the AirProbe Android application collecting from samples from corresponding sensor boxes.

One important feature of the AirProbe module is the calculation of a black carbon (BC) value which is not observed directly. The BC value is derived from a variety of sensor readings originating from the sensor box. These sensor readings include NO_2, CO, VOC, O_3, humidity and temperature. In order to derive the BC value from these readings, a calibration model based on neural networks has been trained as further reported in Deliverable 1.2. This model represents a powerful learning approach and helps to further increase accuracy over time as more and more samples are recorded and aligned to BC values recorded by highly accurate reference meters. The *AirProbe* module utilizes this model to enrich the *AirProbe* data it receives with BC values.

Note that the calibration model is very location dependent. A separate module had to be trained

for each city participating in our large scale case studies. Thus, it is important to associate location information with each of the recorded samples. Unfortunately, the sensor box as well as the smartphone might not be able to provide such a location. The *AirProbe* module, just like a few other modules (e.g. the *WideNoise Plus* module), provide the functionality to infer locations based on IP addresses. Since data is usually sent live via an internet connection, each sample is associated with an IP address. Consequently for each sample which does not contain location information, the module will contact a provider which maps IPs to locations and fills in the missing location information of a sample.

Batch Components

As mentioned earlier the batch components of the data processor are responsible to calculate further higher level semantic information. This includes for example grids and clusters or heatmaps. Those are time consuming to calculate. There are two main components which will be introduced in this section: the cluster component and the heatmap component.

Cluster Component One of the main features of the web interface is the measurement visualization on a map. In order to motivate the users to explore the collected data and share more of their own, it is essential to provide the best possible user experience when browsing on the map. Aggregation into clusters and grid cells representing higher level semantic information is one way to allow a quick overview on large amounts of data at first glance. Since data is coming in constantly clusters and grids keep evolving. Thus, the discovered clusters will have to be recomputed or already existing clusters need to be updated depending on the learning approach used for clustering.

The clusters are computed on the server and only the aggregated cluster information is transmitted to the client in order to keep things as responsive as possible. However, the aggregation has to be computed for each request (for different viewports, zoom levels, etc.) separately. This is time-consuming, especially as the amount of data increases with continuously recorded measurements.

In order to provide cluster or grid data as fast as possible, our solution is to aggregate data for possible requests beforehand. The aggregated data is stored as a collection of spatial objects. These collections are queried instead of calculating aggregations for each request on the fly.

This spatial caching mechanism is outlined in Figure Figure 2.4. For an incoming request, first a spatial object collection is selected based on certain meta attributes. Then the spatial objects within the specified longitude-latitude bounds are retrieved.

Spatial objects are stored in the data table. Each object is described by a collection id, an object id, a longitude-latitude pair and some object specific information such as sensor values or associated tags. The meta attributes of a spatial object collection are stored in the meta table and are used to retrieve the stored collections. These attributes are:

- *Data type*: Distinguishes collections of different nature.
- *Spatial type*: Distinguishes collections by their spatial properties like clusters, grids or tracks.
- *Sub type*: Distinguishes sub-collections representing the same samples but different corresponding data. This allows to dynamically load additional information about spatial objects after transmitting the initial data given an object id.
- *Zoom level*: Spatial objects are aggregated values. Those values are aggregated differently for each zoom level defined by the map.

Using this spatial object cache we are now running batch jobs on our data in order to update the spatial collections regularly. The rendering time of large clusters was reduced from several minutes

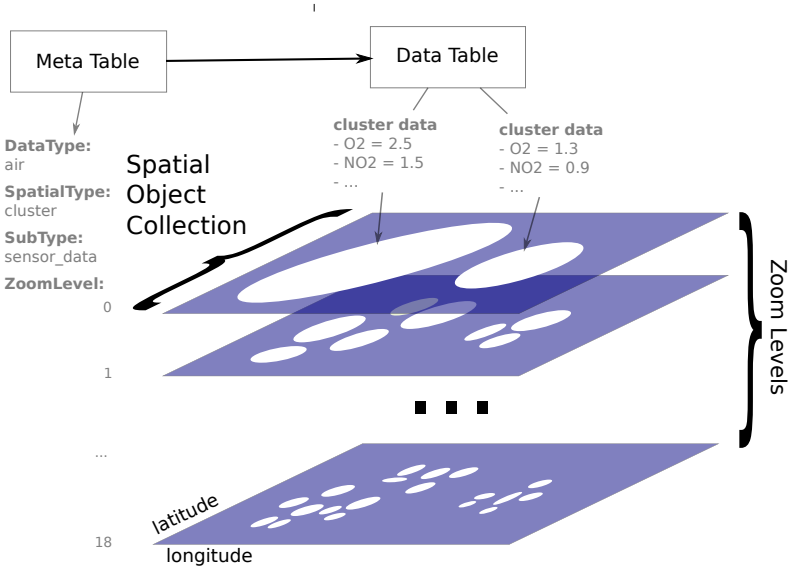


Figure 2.4: Outline of the spatial object cache concept.

to a fraction of a second. This enabled a smooth visualization of large amounts of data on the map. Currently we are running an incremental clustering algorithm, thus, clusters are also updated in real-time.

Heatmap Component We are using heatmaps to provide a collective view on the air quality data recorded by the *AirProbe* application Figure 4.4. A custom interpolation scheme based on radial basis functions is used in order to account for missing data resulting in a smooth visualization of the collected samples. The heatmap component calculates image tiles needed for the OpenLayers visualization (see Chapter 3 for more information) for each zoom level by first aggregating samples into a grid of pixels and the applying the interpolation method to generate a smooth surface. The result is served to the public as well as individual users via a Tile Map Service (TMS).

Chapter 3

REST API

The API provided by the *EveryAware* platform uses the REST protocol (representational state transfer), which is widely adopted by Internet of Things platforms like Xively[1] or ThingSpeak[2]. Thus, we focused on REST in order to make our API as simple as possible for our users.

The REST API has been largely redesigned since the last deliverable to support more flexible usage and a clearer structure (though the API from the last deliverable is still mostly available). There are three main components of the general API: the data endpoint for receiving packets of data, the endpoints to retrieve data points themselves and the endpoints for retrieving statistics and higher level information derived from the data points. See Figure 3.1 for the resource paths.

```
(POST) /api/v1/packets
(GET) /api/v1/data/**
(GET) /api/v1/statistics/**
```

Figure 3.1: Basic resources of the REST API.

The `packets` endpoint allows to write data to the system. The posted content can be any textual input. There are several meta-attributes which are supported by the *EveryAware* system natively in order to support data processing and controlled querying without understanding the content itself. Meta-attributes include, but are limited to: data type and format, recording time, location data, or feed and visibility information.

The `data` endpoint enables access to the data. There are several query parameters based on the meta-attributes which can narrow down the data selection. The most general `data` endpoint is the packet endpoint which allows to access any textual input sent to the `packets` endpoint. Also, there are several specific data types supported natively: *WideNoise Plus* data which represents data sent by the *WideNoise Plus* application, or *AirProbe* data which data points sent by the *AirProbe* application. See Figure 3.2 for examples.

```
(GET) /api/v1/data/packet
(GET) /api/v1/data/widenoise
(GET) /api/v1/data/airprobe
```

Figure 3.2: Data endpoints for accessing packet, *WideNoise Plus* and *AirProbe* data points.

The `statistics` endpoint is used to serve statistics and higher level semantic data based on the data sent to the `packets` endpoint. Some examples are the number of active devices or

[1] `http://www.xively.com`
[2] `http://thingspeak.com`

summaries of user sessions. See Figure 3.3 for examples.

```
(GET) /api/v1/statistics/data/airprobe/activedevices
(GET) /api/v1/statistics/sessions
```

Figure 3.3: Statistics endpoints for accessing currently active *AirProbe* devices and for accessing user session information.

An important feature of the API is the security policy. We have adopted the OAUTH2[3] protocol for authentication and authorization. Thus, calls to the API are authorized by using so called access tokens. These tokens can be acquired via the `token` endpoint (see Figure 3.4).

```
/oauth/token
```

Figure 3.4: Endpoint to acquire OAUTH2 access token.

The smartphone applications are using these endpoints to communicate with the *EveryAware* platform. They are first acquiring an OAUTH2 access token. Then they send data via the `packets` endpoints. And finally the request uploaded data and statistics from the `data` and `statistics` endpoints.

In addition to the REST API we are also providing KML[4] data as well as Web Map Services[5] (WMS) back to the community. This includes public aggregates as well as private, user-centric aggregates. This includes, but is not limited to, clustered and gridded data from *WideNoise Plus* and *AirProbe* in KML format as well as heatmaps of the collectively recorded air quality data from *AirProbe*. See Figure 3.5 for corresponding URLs.

```
/files/public/airprobe/heatmap/bc/tiles
/event/widenoise/api/data/air/cluster/kml
```

Figure 3.5: URL to access TMS (a variant of WMS) formatted heatmap tiles of collectively recorded air quality data from *AirProbe* and a URL to query KML formatted clusters of *WideNoise Plus* data.

Besides the mentioned filtered data access possibilities and the aggregated we offer a nightly dump of the data to the Consortium members[6].

[3] `http://tools.ietf.org/html/rfc6749`
[4] `https://developers.google.com/kml/documentation/`
[5] `http://www.opengeospatial.org/standards/wms`
[6] `https://www.kde.cs.uni-kassel.de/datasets/ubicon`

Chapter 4

Web Interface

There are two major applications which have been implemented on our platform: *WideNoise Plus* and *AirProbe* [Becker et al., 2013b]. Both have been developed as part of the *EveryAware* research project.

WideNoise Plus is an application for monitoring noise pollution and *AirProbe* monitors air quality. The data collected by the smartphones are transmits to our REST API where it is augmented and aggregated by the *data processor* to be visualized on the web interface.

One important feature of the web interface is its integration with social services like Twitter[1] and Facebook[2]. It is achieved by using the Spring Social module as mentioned in Chapter 2 and also by utilizing several plugins provided by Twitter and Facebook themselves. Figure 4.1(a) and Figure 4.1(b) show an example of this integration. When finding an interesting place on the map, it is possible to share this place on Twitter and Facebook. Furthermore, we are connecting objective data with social context by incorporating filtered social content from social networks into our system. An example can be seen in Figure 4.2 where we add relevant Tweets to the *AirProbe* air quality map.

4.1 AirProbe

AirProbe is a system for collaborative air quality monitoring. The sensor box produces readings from several air quality related sensors such as NO_2, CO, O_3, VOC, temperature, and humidity. Users may add tags concerning their measurements adding semantics or subjective information to the otherwise objective measurements.

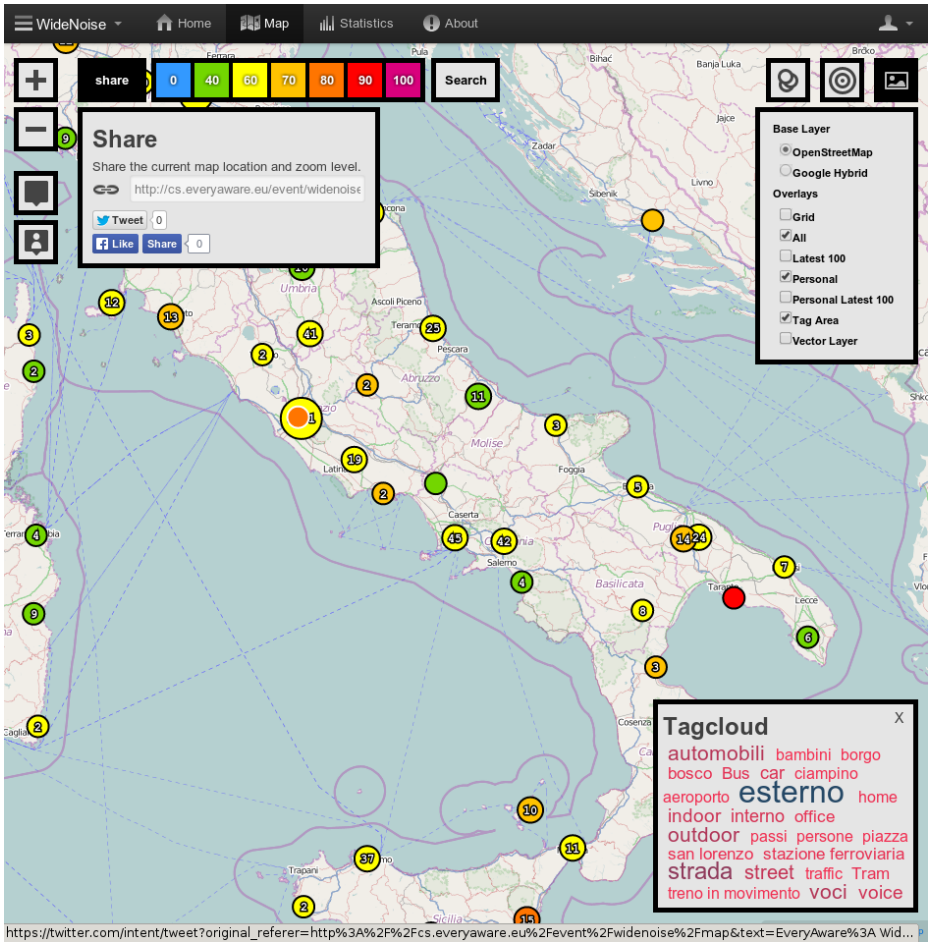
The web application provides several views on the data summarizing it in various ways including a map with different layers as well as several global and personal statistics.

The map visualizes the collected data on a map which allows for an easy access to the data as well as for obtaining first insights. The OpenStreetMap[3] map provides a quantitative view on the data by aggregating samples using clusters, grid as well as a heatmap view in order to emphasize the covered area as shown in Figure 4.3 and Figure 4.4. Both views are also available for individual users summarizing their personal data.

The map view also supports the active tracking of currently measured data. Tracking these current measurements is further supported by providing data compliant with Google Earth for 3D visualizations.

The statistics calculated by the *AirProbe* application include global as well as personal statistics like latest overall measurement activity or air quality averages. Furthermore, they list user sessions,

[1] https://twitter.com
[2] https://facebook.com
[3] http://openstreetmap.org/



(a) A screenshot of the *map page* of *WideNoise Plus*. In the upper-left corner the share buttons for Twitter and Facebook are shown.

(b) A screenshot of a shared map fragment on Twitter.

Figure 4.1: *EveryAware* functionality for sharing content on Twitter.

Figure 4.2: Adding filtered social content (Tweets) to objective data (*AirProbe* air quality measurements).

give short summaries regarding those sessions enable the user to view and replay those session. The personal sessions overview can be seen in Figure 4.5(a). One view for exploring personal sessions can be seen in Figure 4.5(b).

The *AirProbe* module of the *EveryAware* platform also had to support the APIC case study. The case study was held in order to gather large amounts of air quality samples and behavioral shift patterns using the sensorboxes in the four cities Antwerp, Kassel, London and Turin. In order to keep the motivation and competitiveness as high as possible for the teams playing, we implemented a ranking mechanism balancing repetitive sampling and coverage: The map was divided into 10 by 10 meter grids. One point was given to the team when sampling within one such grid cell. When the team received a point in a particular cell, the player did not receive a point from this grid cell for half an hour. The results for each city as well as for each team have been visualized and updated in regular intervals on the *AirProbe* website as can be seen in Figure 4.6. Figure 4.6(a) shows the ranking of each city visualizing the coverage and providing several statistics. Figure 4.6(b) shows a detailed view of the point-coverage of the city, in this case Kassel.

4.2 WideNoise Plus

The *WideNoise Plus* web application aggregates, summarizes, and illustrates noise-related data collected by the *WideNoise Plus* smartphone application [Becker et al., 2013b]. This smartphone application is recording environmental noise levels and allows the user to express certain perceptions about the recorded samples via perception sliders (e.g., Love–Hate). To further characterize the samples it is possible to attach custom tags. In order to share samples with friends or the general public the smartphone application also supports posting results on social media.

The *WideNoise Plus* web application provides several data summarization views including the map

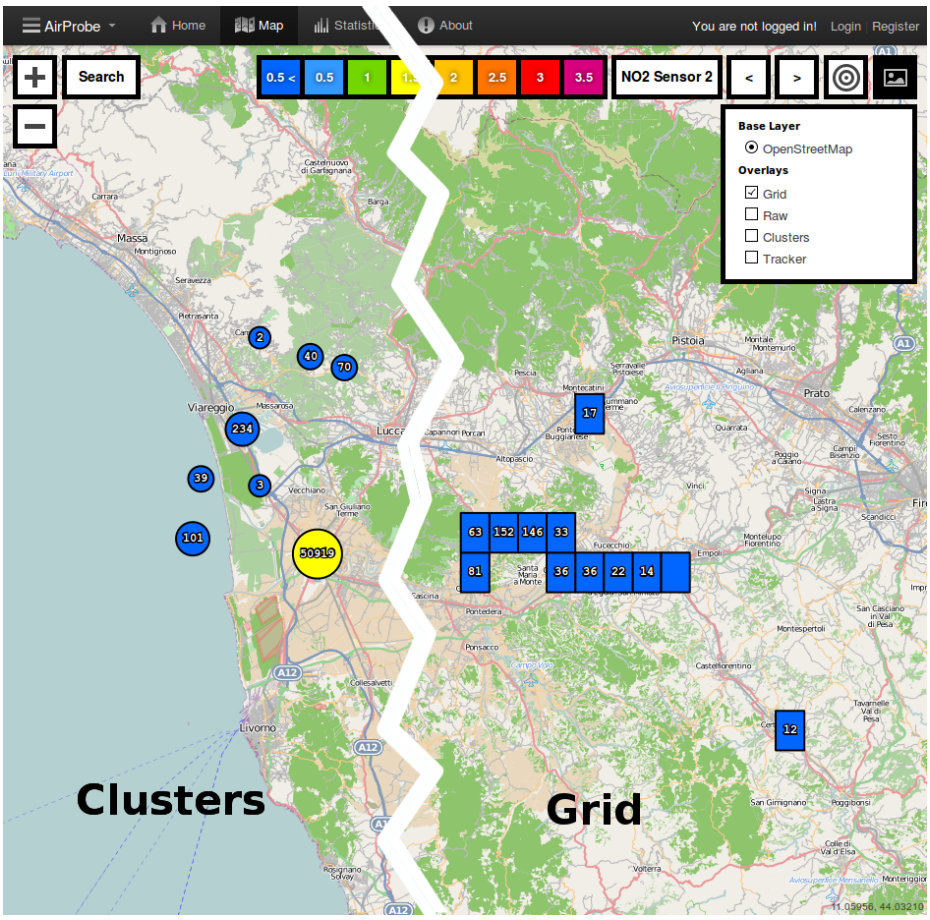


Figure 4.3: A screenshot of the *map page* of *AirProbe*. The left side shows the cluster view, the right side shows the grid view.

and several statistics pages. The map is shown in Figure Figure 4.8. It displays, for example, a clustered view on global and user specific measurements (providing corresponding detail information on demand [Shneiderman, 1996]) or a tag cloud characterizing the summarized data by its semantic context.

The user can access several live statistics about the collected data allowing to trace current measurement trends or get an overview of the collected data. Some of these statistics are:

- Number of measurements users collected in the past two weeks.
- Activity of users for each continent during the past three days.
- Latest recordings and recent average values for different time intervals and locations.
- User rankings including users with most samples, the most active users, etc.
- Tag clouds characterizing the semantic context of the measurements.

Figure 4.7(a) and Figures Figure 4.7(b) show the global and personal statistics pages.

A second type of statistics can be accessed by users via their personal page for information on their own measuring behavior. As an alternative to the map visualization, the page also provides a KML[4] (Keyhole Markup Language) export containing all the user's measurements.

[4] `http://opengeospatial.org/standards/kml/`

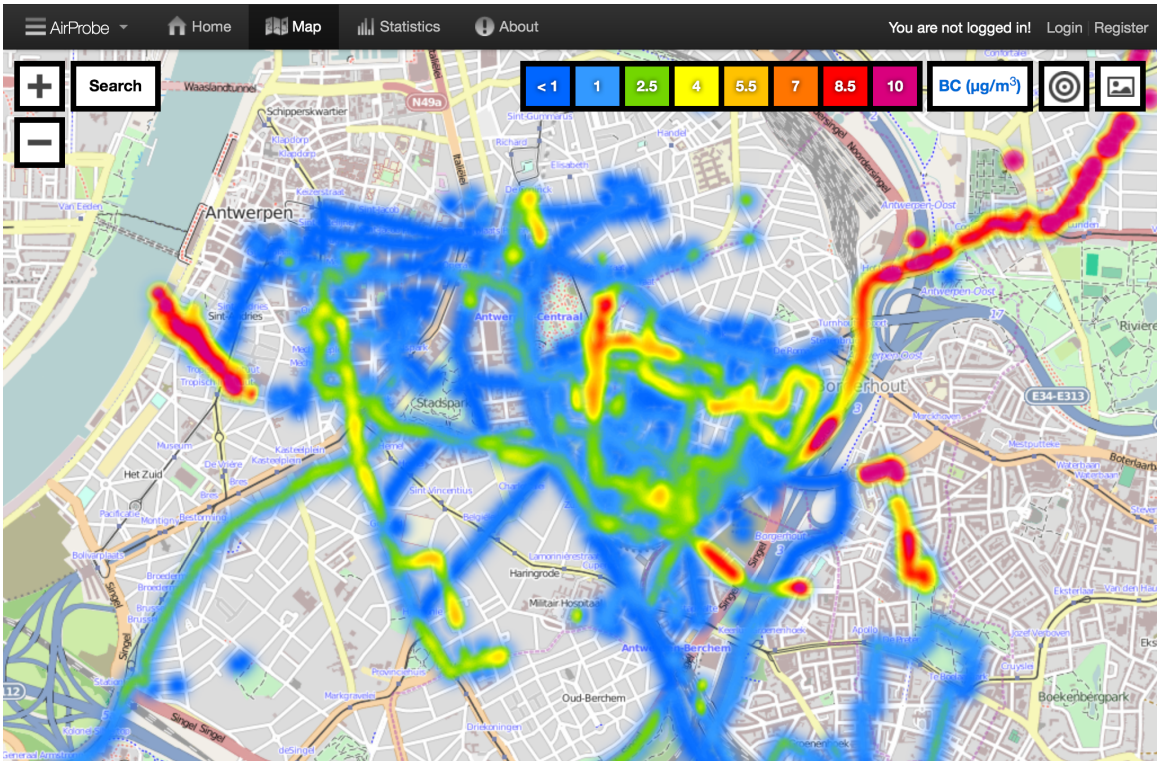
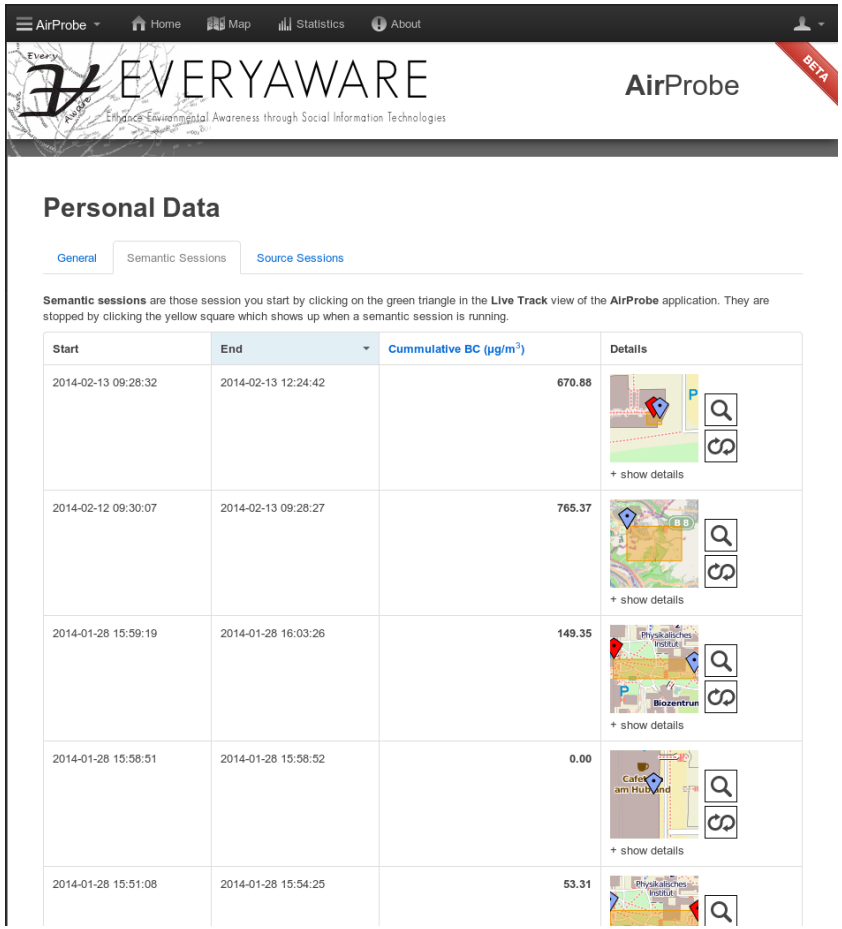
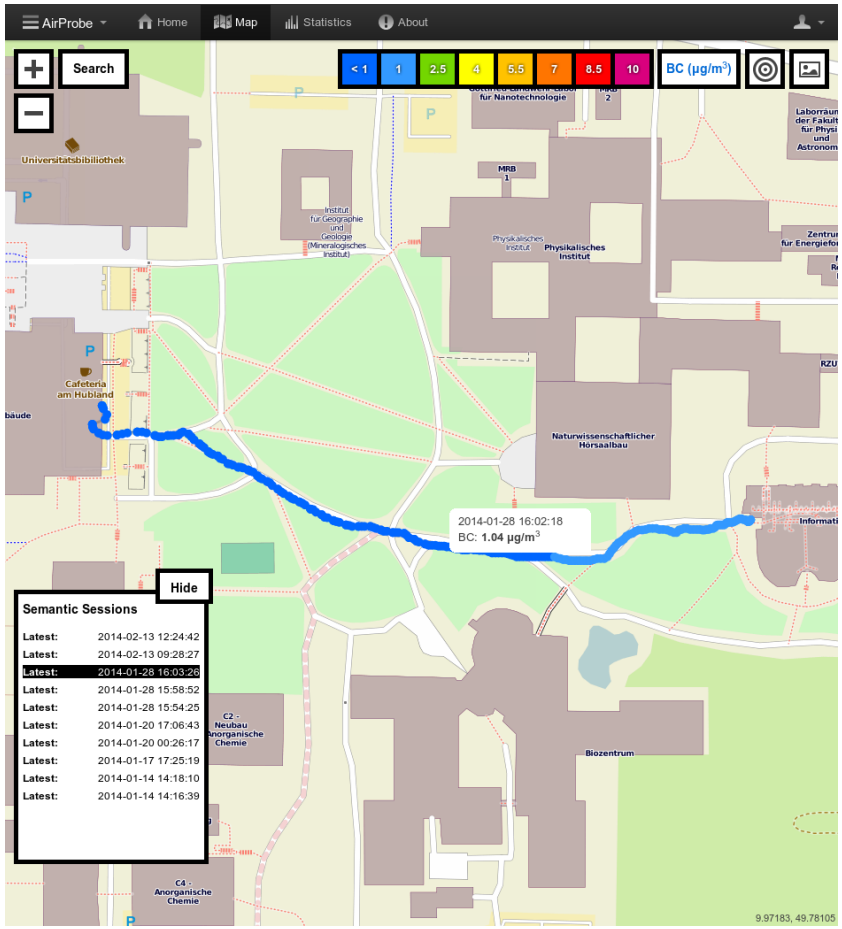


Figure 4.4: A screenshot of heatmap on the *map page* of *AirProbe*.

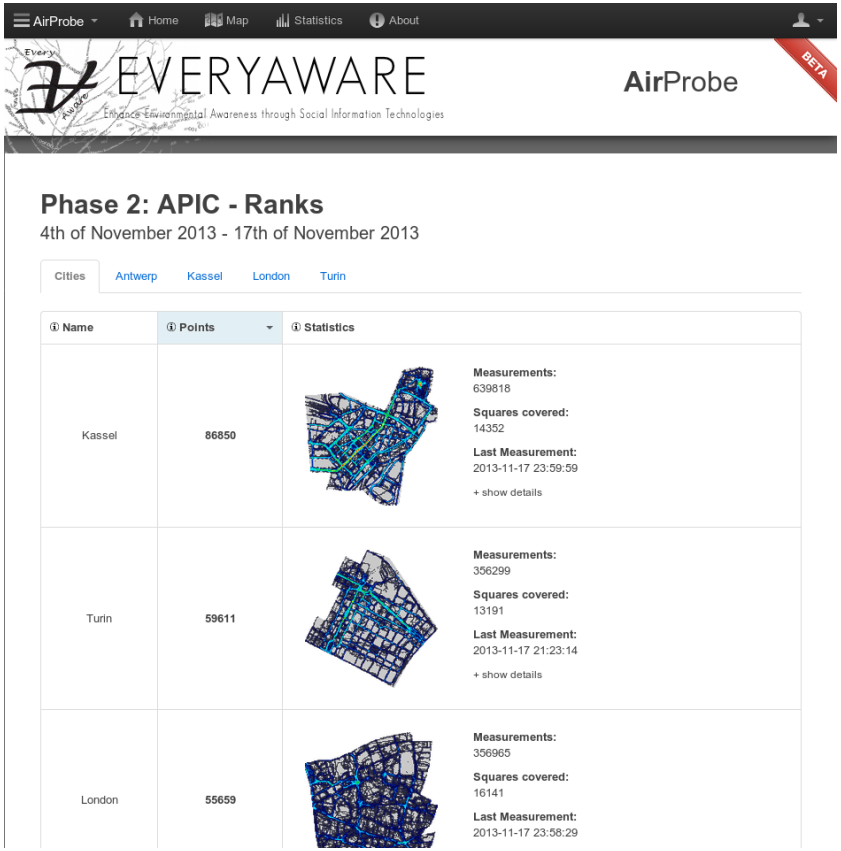


(a) This *AirProbe* view shows a user's personal sessions.

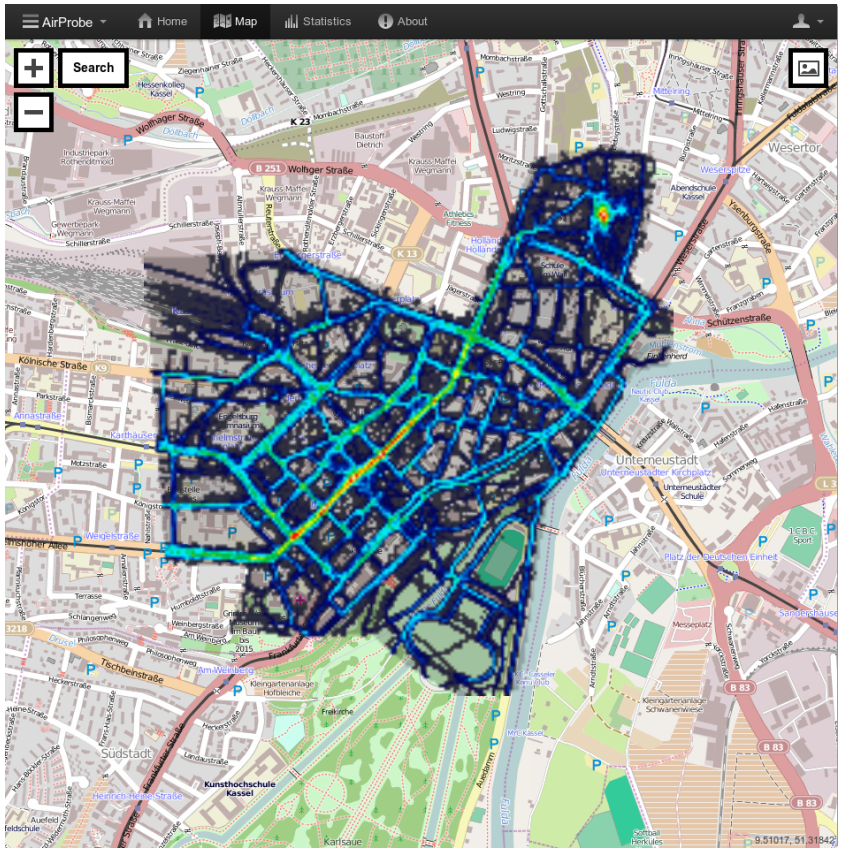


(b) This *AirProbe* view shows a view for exploring individual user sessions.

Figure 4.5: Personal session visualizations.



(a) APIC city ranking.



(b) APIC point coverage for Kassel.

Figure 4.6: APIC ranking visualizations.

(a) The global *WideNoise Plus* statistics page.

(b) The personal *WideNoise Plus* statistics page.

Figure 4.7: *WideNoise Plus* statistics pages.

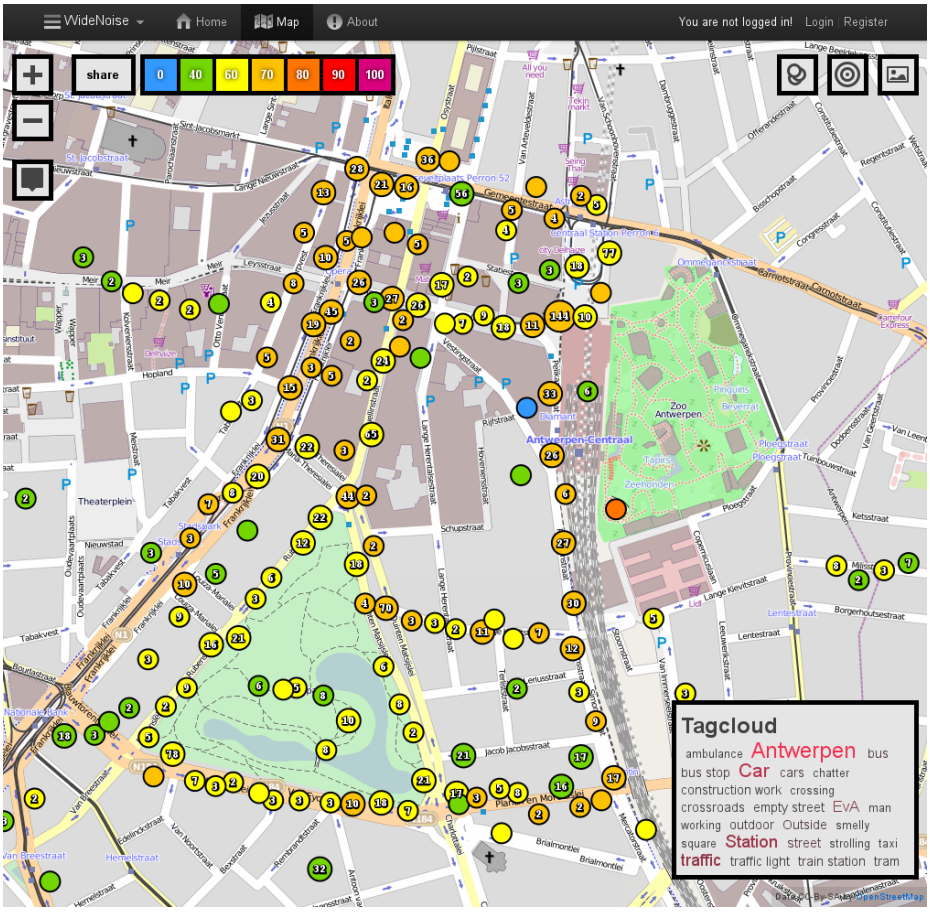


Figure 4.8: Screenshot of the *WideNoise Plus* map page.

Chapter 5

The Experimental Tribe gaming platform

The Experimental Tribe gaming platform [Caminiti et al., 2013; Cicali et al., 2011] (XTribe in the following) was planned to be already operational in the second part of the project. The platform has been used to implement and host the web-based experiment used in connection with the main large scale case study of the project, the AirProbe International Challenge (APIC). Therefore, our efforts in terms of development were focused mainly on testing its response and stability to large user access.

5.1 Large scale experiment testing

We developed experiments with the aim to reach a large user participation. In particular, we designed an put in action the *laPENSOcosi* experiment which, beside its intrinsic relevance, also proved that XTribe could withstand thousands of users and hundreds of simultaneous accesses. *laPENSOcosi* was a web experiment in the shape of an *entertaining personality test*, where we explicitly investigated users opinions on political entities (parties, coalitions, individual candidates) of the Italian political scene. The aim of the experiment was to study the political perception of italian voters before the political elections of February 2013. We found that when ranking candidates according to the user expressed preferences, the perceived distance of the user from a certain candidate scales as the logarithm of this rank, similar to a Weber-Fechner law. A detailed description of the result of the experiment is beyond the scope of this report. What matters here, is that the platform granted its service in such a large scale experiment. We released *laPENSOcosi* on the XTribe portal on the middle of January 2013. At the early days of March 2013 (elections were on 24-25 February 2013) the experiment gathered 81508 opinions expressed by 1727 users. Thus we had an average of more than 1600 opinions. Moreover, we registred peaks of almost 10000 opinions from more than 200 users in a single day without any trouble for the platform.

5.2 Integration with other platforms

The impressive results in terms of participation obtained by the *laPENSOcosi* experiment, were made possible not only thanks to traditional advertising methods (through newspapers, news website, radio, etc.) but also exploiting the XTribe Facebook integration capabilities. Actually, thanks to Facebook developer functionalities and some apposite Drupal modules (XTribe is Drupal-based), users are able to:

- register on the platform and login in one click through their Facebook account;
- join to XTribe experiments within Facebook, by playing with the XTribe Facebook app;

- invite their friends to play on the platform;

- share results of their game experiences on Facebook through the Facebook API.

Another important integration implemented was with the Mechanical Turk recruitment. XTribe can be used in conjunction with the Amazon Mechanical Turk platform (`www.mturk.com`, a virtual marketplace for small online jobs such as image annotation, polls, translations, etc; will be referred to as AMT in the following) in order to exploit its ability to recruit users with a modest monetary investment. AMT can be used to enhance participation and possibly in the initial phase of an experiment, to provide the necessary pool of data to begin with. In particular, we used AMT users in the City Race experiment described in the Deliverable 2.1. For this analysis, we discuss two user groups, corresponding to two experimental settings with different stress levels and goals, who show differences in performance. One group was composed mostly by players taking part in a special event organized in a bookstore in Rome already described in Deliverable 6.2, while the other was set up by recruiting players in the virtual labour market of AMT. Through these experience we proved that XTribe can be successfully used as a host for complex AMT tasks and that researcher can safely recruit users on the Amazon platform when needed. A possible integration strategy, the one used for CityRace, is to provide a payment code at the end of the game for AMT users, as is portrayed in Fig. 5.1.

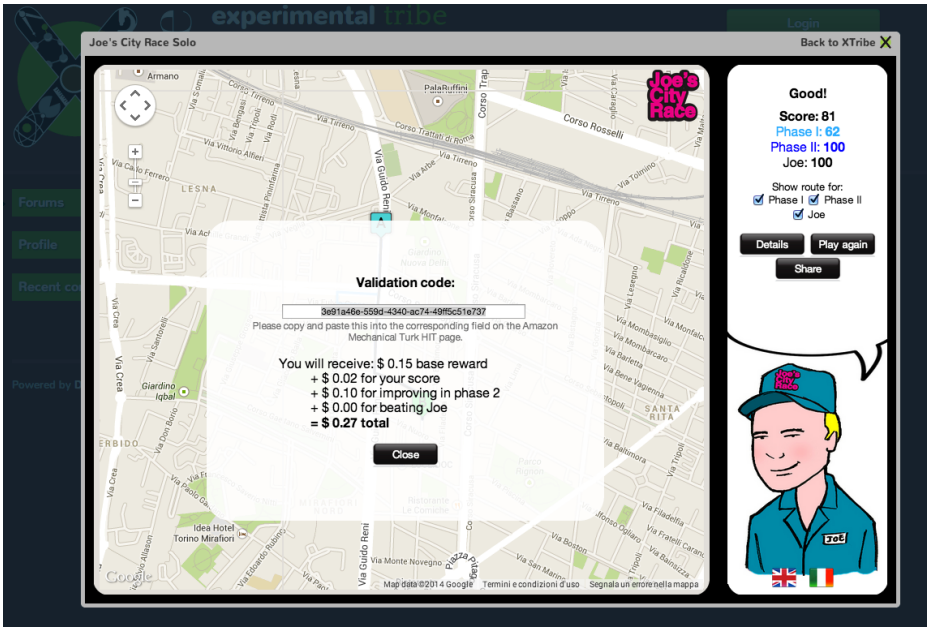


Figure 5.1: Screenshots of the CityRace interface, during the AMT code retrieval.

Beside this, we also invested time in promoting the platform by participating to international conferences in order to reach one of the main aim of the XTribe: the creation of a community of developer. At present time, there are several developers active on the platform working on their own experiments. Thus we implemented a set of tools to support the community such as a forum integrated on the platform and a documentation portal (`http://doc.xtribe.eu`) where guides, tutorials and XTribe API documentation can be found.

5.3 The AirProbe International Challenge web game

5.3.1 Introduction

The AirProbe web-game has been developed to gather and monitor citizens' subjective opinions about air pollution during the final case study of the EveryAware project, named **AirProbe International Challenge**, APIC in the following.

5.3.2 The challenge

APIC was presented as a competition between four cities: Antwerp (Belgium), Kassel (Germany), London (England) and Turin (Italy). The aim of the competition was to build a map of air pollution for each city. People from each city willing to join could become either **Air Ambassadors**, to measure the levels of air pollution with the sensor box developed by the EveryAware Consortium, or **Air Guardians**, to report subjective air pollution level estimations in various spots of their city. The challenge was divided in three phases, each of them lasting two weeks, and in each phase the two kinds of volunteers had to perform different tasks.

The Air Ambassadors had the assignment to cover a given area measuring air quality levels with the provided sensor box. While Air Ambassadors had to play the web game and to recruit players in phase one, their measuring activity started in phase two and had to be fairly distributed in space and time, i.e., trying to optimize space-time coverage. Instead, during phase three, Ambassador were free to measure wherever they liked. The specific tasks for Air Guardians were actually virtual tasks in the web-game context and will be described in the following.

5.4 Game Concept

In order to gather subjective opinions about the air pollution in the four cities we decided to follow the GWAP approach and accomplish the task using a web-game. We started designing the game taking inspiration from the peculiar kind of data we wanted. Our specific aim was not only to get a map of perceived air pollution but also to study how the perception is affected by objective data. In fact, the case study specifics foresaw volunteers opinions monitoring before, during and after exposition to objective pollution data, obtained by AirAmbassadors sensor-boxes. So we needed to keep players engaged in the game for the longest time possible, in order to monitor the opinion shift of each player. Beside this, opinions about air pollution had to be geolocated so the game had to take place on the map of the four city. In particular, for each city we defined an area of data gathering of approximately 3 square kilometers. The areas for each city are represented in Fig. 5.2.

The most suitable kind of game seems thus to be a management simulation, like the famous FarmVille or Harvest Moon. In this kind of game the user have the task to take care of a given territory. By improving his management performances, the user increase the income in the game virtual currency. Thus he may access a wider set of interaction, for example he can expand his territory or buy more stuff, trying to get a further improvement. The periodic rhythm of this vicious, or virtuous circle, is marked (in FarmVille-like game) by the ripening time of the income: in order to generate a revenue, each action required a given time, spanning from few seconds to several hours. This mechanism is an incentive to return to the game, in order to gather the results of the efforts.

The AirProbe web game is a simplified kind of map management game. We reported in Fig. 5.3 the interface of the game. Players are called to fulfill their role of Air Guardians by annotating the map with AirPins, geolocated flags with an estimation of the pollution level identified as the level of Black Carbon in $\mu g/m^3$ within a scale from 0 to 10. The game area of each city is divided in

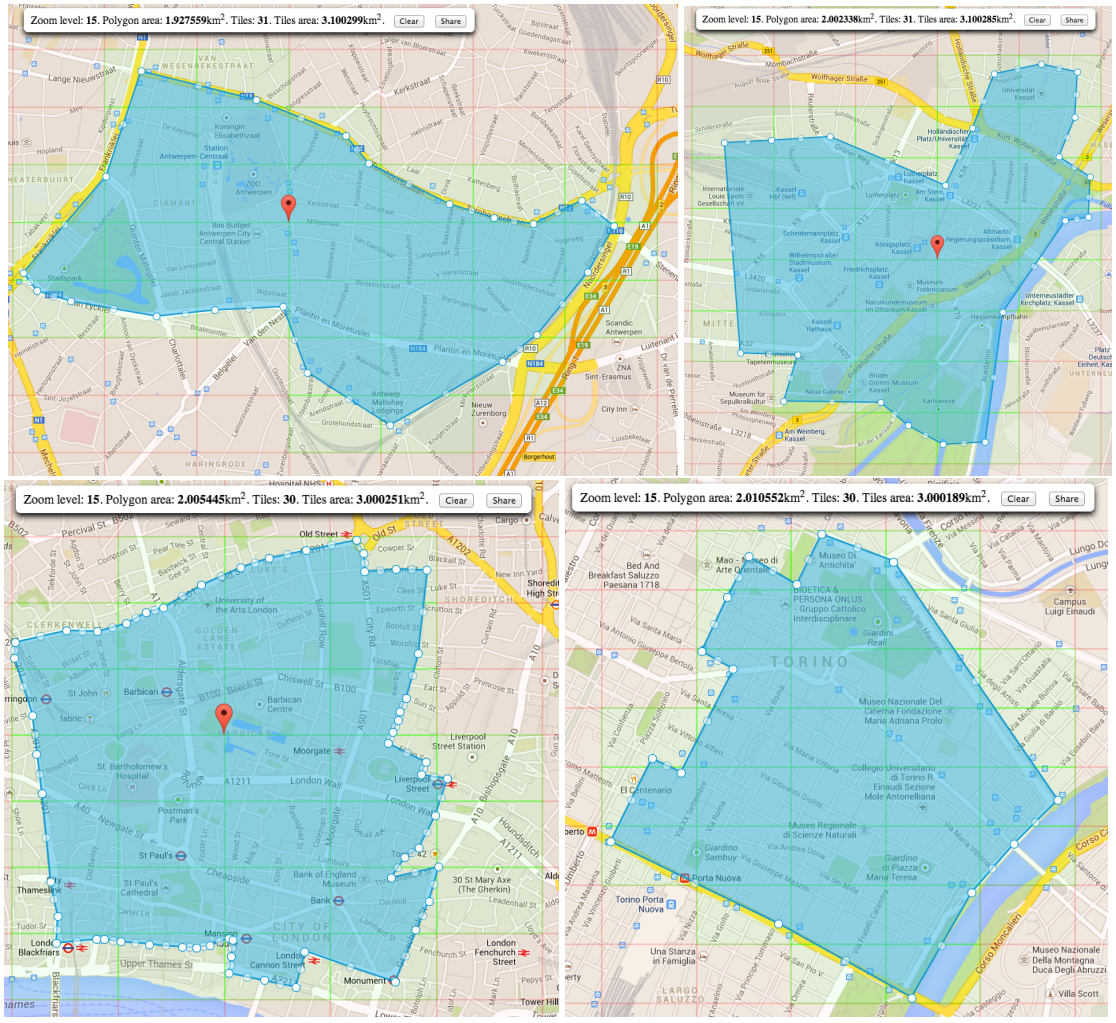


Figure 5.2: In green the game areas and in blue the measurements areas for the four cities.The grid represents the tiles division for the web game. From the top left to the bottom right: Antwerp, Kassel, London and Turin.

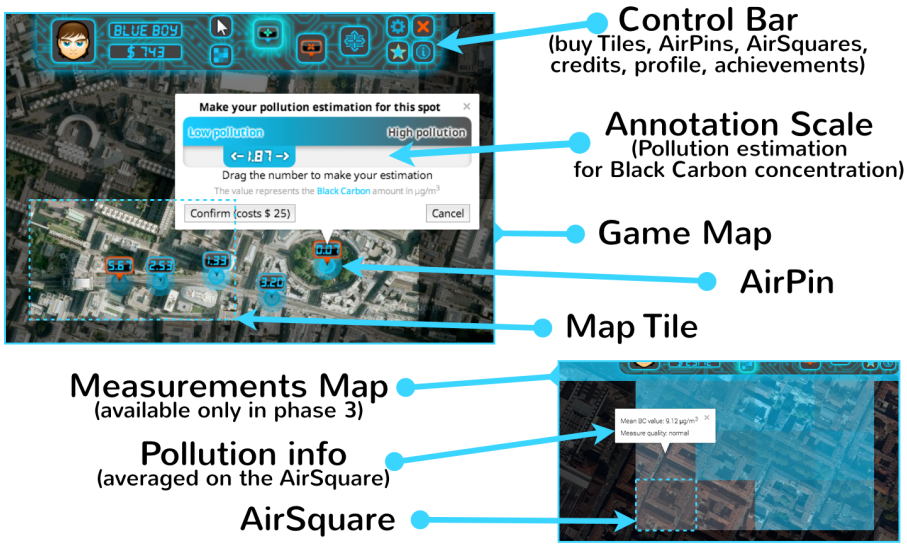


Figure 5.3: Screenshots of the game interface, with indication of the main entity and tools.

Tiles as indicated in Fig. 5.2. At the beginning of the game, users are asked to create a profile (by choosing an avatar and a name) and to choose a city and a team. Teams were linked to Air Ambassadors, and were an important part of the competition. Then the volunteer starts from a given Tile of the map of the chosen city. The user can interact by placing (or editing or removing) AirPins or by expanding his territory by buying more Tiles. Each day the AirPins placed generate a revenue calculated on the precision of the annotation (more details in the following). In order to collect the revenue generated every day by each AirPin, the user has to access daily, otherwise the revenue will be wasted. The revenue collected will be added to the user balance, and can be used to buy more AirPins and more Tiles, and so on. In order to improve motivation and fidelity, there are bonus for day-in-a-row accesses and a great set of achievements. These achievements consisted in prizes at given milestone in the game story: a certain numbers of AirPins, or of Tiles, or for the precision in the annotation, and so on. In third phase of the case study we made available information about objective measurements gathered with Air Ambassador sensor box. We avoided to give punctual information about measurement, otherwise it was likely that users would simply copy the punctual values. So we decided to release aggregated information by introducing a new map partition named AirSquares. Each Tile contains 12 AirSquares, that can be purchased just like AirPins or Tiles. Once the user bought an AirSquare, he can se the average pollution value in that area, so the task become to estimate fluctuations.

5.4.1 Revenue and feedback

Our case study, as we said, was divided in three phases. Beside the AirSquare introduction in phase three, the only change between phases affected the revenue calculation algorithm. We generically said that revenue was related to precision of the annotation. Let us now define the meaning of 'precision' in our context. In the first phase there were no objective data for comparison, thus we adopted the strategy of matching the AirPins value with the value of other users AirPins within a certain range (30 meters). The general algorithm of revenue calculation for a certain AirPin X of coordinate (X_{lat}, X_{lon}) and value X_{BC} was chosen in order to fulfill these conditions:

No data Even if we solved the problem of the lack of data by comparing a user annotation with other users annotations, at the beginning of phase one also other users annotation were

missing. So, in case of absence of other AirPins within the range, the only choice was to trust the user and give him an average revenue for the AirPin.

Distance In case of other AirPins within the range, their distances from X location had to be taken into the account.

Reliability of the match A match with a greater number of others AirPins had to correspond to a greater revenue. So the maximum possible revenue is determined by the number of AirPins within the range.

We decided that the most simple and reasonable choice was a confrontation with the average of all the AirPins (including X itself) within the range, weighted with function of the distance from X location, and rescaled depending on the number of annotation. So let F be the set of all flags within 30 meters from X, including X itself and consider the sequence $(F_i(BC), F_i(Dist(X)))$ of Black Carbon estimations values and distances from X of all flags in F. Let P_{BC} be the weighted mean of F_{BC} using weight $W_i$defined as

$$W_i = 1 - F_i(Dist(X))^2/30^2 \tag{5.1}$$

Let W be the sum of all W_i. We computed the maximum revenue based on how many flags are there, more precisely it is based on W. We use an inverse exponential function to smooth the max revenue (MR) from 30 (when $W = 1$) to about 65 (when $W = 10$) to 75 (when $W > 20$):

$$MR = 30 + 45(1 - 2^{-(W-1)/3}) \tag{5.2}$$

We now define the 'error' E_X of the estimation for the X AirPin as the absolute value of the difference between X_{BC} and P_{BC}. Finally, we defined a critical threshold C for the error. If $E_X > C$ then the revenue will be 0 anyway otherwise the following formula is used to calculate the final revenue R_X for X:

$$R_X = MR(1 - E_X/C) \tag{5.3}$$

Users perception of the revenue was collective: they only know the cumulative value for the whole ensemble of their AirPins. The only feedback regarding single flags is a red sign for flag that are not generating any revenue.

As we said, the revenue algorithm has been different in each of the three phases:

Phase one The threshold for the error was very tolerant ($5\mu g/m^3$) in order to make the game easy at the beginning.

Phase two The threshold was made smaller ($2.5\mu g/m^3$), in order to push users to improve their performances.

Phase three The threshold was unchanged but real measurements from Air Ambassadors sensor boxes were used instead of other players annotation to calculate the revenues.

Users were not informed about the detail of the algorithm. They were just told to try to be precise.

5.4.2 Design Issues and possible future solutions

No punctual feedback As we said, a direct feedback about the revenue generated by each AirPin was not given to avoid users to simply adapt the flags value in order to optimize the local revenue. This has caused a lot of complaining from the users. In fact, this partial blindness about the consequences of the game interaction is quite frustrating. A possible solution is to make available direct feedback only by paying a certain (high) amount of virtual credits and only for a limited number of AirPins.

Scale meaning At the beginning of the game, many users did not knew nothing about the scale meaning. In order to make them use the range in a realistic way an histogram of real measurement above the city may be represented near the scale. In this way users may understand better what values have to be used and what values are unreasonable. Another way to act is to avoid completely the issue by presenting them just a perceptive scale, unbounded from real values.

Prices, revenues and scaling In the actual formulation the game was slightly too easy and pushed players to expand their territories and place more flag, instead of trying to fine tune what they have. In this way their maps grew larger then expected and the game interfaces were overloaded, in some cases with hundreds of flags. A possible solution would consist in raise the prices. Even if this will probably let players focus a little more on the quality of their flags instead of the quantity, this is not resolutive because the overloading process will simply slow down but will still be there, at a certain point. Another possible solution is to make AirPins decay after a given amount of time. In this way users are forced to come back and to annotate places several time. The decay may also not be fixed but caused by other factor, such as virtual random disaster ("Sorry, a flood destroyed some of your AirPins") or attack from other users (see the following point).

Poor social interaction Even if the game is integrated with social networks and allow users to share contents, this point can be greatly improved. It could be interesting, for example, to allow volunteers to check the annotation of their friends, or to attack the friends, destroying some of the AirPins. This would also be a possibility of enrichment of the interface: a market of attack/defence gadget would be introduced, making the game more appealing.

5.4.3 Ranks and prizes

Every day were compiled ranks about the players performances. In order to boost motivation, we introduced a set of prizes to be given at the end of each phase and in each city. We considered two main metrics for the ranks: the total revenue of the last day of play and the number of day played in each phase (we will call it fidelity). The basic prizes scheme was the following: top five last day total revenue were awarded with t-shirts, while the best last revenue got a backpack. We also assigned another backpack to the most faithful user and some t-shirts to the other top fidelity players. In case of ex aequo (which for this quantity are not so unlikely) average precision in the annotation was used to decide the winners (precision is defined as the average revenue per flag, calculated on a minimum of 20 flags). In Turin the basic prizes scheme was strictly complied. In Kassel and London the scheme was enriched with some Amazon voucher to best teams for few hundreds of euros / pounds. In particular each team received €50 / £100 in Amazon vouchers and the team with the best time/space coverage and the largest number of active Air Guardian players won €250 / £400 in Amazon vouchers. In Antwerp the prize scheme was much more flexible and has been adapted to the low number of players.

5.5 Implementation details

The game's user interface has been developed using standard web technologies. It consists of about 700 lines of HTML 5 and CSS 3 together with almost 3000 lines of JavaScript code organized in several objects. The core object is in charge for handling all client-server communication trough XTribe, coordinating all other objects, and maintain the game status.

The game makes extensive use of Google Maps API 3.0 and there is a MapManager object in charge for all the map related interactions. The most noticeably Google Maps features used by the

game are: configuration and customization, event listening, custom markers, coordinate conversion, drawing API, map movement and zooming functions, pan constraining, custom UI controls. This latter feature required a dedicated JavaScript object: UIManager. It handles all the custom UI controls that are shown on top of the Google Map such as the top control bar, custom messages and popup boxes, etc. These three main objects are complemented by a set of others utility objects for language localization, achievements handling, in-game tutorial, and integration with social networks (Facebook and Twitter).

The server side game manager is written in nodeJS in order to guarantee a high level of concurrency and properly serve users that are interacting with the game at the same time. Game data are stored into a MySQL database. There are tables for users, their open tiles, their AirPins, and the AirSquares they bought. Moreover there is an history table where we store all actions done by each user during the game play with timestamps to allow better analysis of user behavior.

The XTribe platform automatically take care of all message exchange with the client UI and the manager during the initial and final phases of the game. But during the main game phase a variety of custom message are exchanged. A summary of the messages sent from the client UI to the manager is as follows (message parameters appear in parenthesis):

- **playNow**. The user is actually going to play now.
- **saveSettings** [city, language, name, avatar, team]. The user changed her settings.
- **setFlag** [airPinId, latitude, longitude, value]. The user added or modified an AirPin.
- **delFlag** [airPinId]. The user removed and AirPin.
- **buyTile** [tileId, price]. The user bought a map tile.
- **buyAirSquare** [airSquareId, price]. The user bought an AirSquare
- **saveAchievements** [achievementId]. The user obtained an achievement.
- **claimAchievement** [achievementId, revenue]. The user claimed the prize associated to an achievement.

A summary of the messages sent from the manager to the client UI is as follows:

- **settings** [city, language, name, avatar, team, tiles, airPins, airSquares, daily revenue]. For returning users, at the beginning of the game the manager send all saved information.
- **tilesAndFlags** [array of tiles, array of AirPins]. For new users, in response to the first **saveSettings** message, the manager generates initial game data (such as a random initial tile) and send them back to the user to start playing.
- **airSquareInfo** [pollution level]. In response to a **buyAirSquare** message from the client, the manager sends back the associated pollution level.

We remark that most of the exchanged messages also carry information about spent money and balance in order to allow server side checks and discard messages from players that are trying to cheat.

5.5.1 Revenue computation

Once a day the revenues for each user AirPin were computed according with Equation 5.3. That formula requires to confront each AirPin of a users against those from all other users within a give radius $r = 30m$. A naive approach would easily result in an algorithm whose execution time grows

quadratically with the overall number of AirPins and that queries the database for the same values over and over. This would ultimately result in overloading the server itself.

In order to avoid this risk and efficiently compute revenues we consider each city as split into columns of width r. In a generic step i we compute the revenue of each AirPin in column i while having in memory also all AirPins in columns $i-1$ and $i+1$. At the end of step i we discard column $i-1$, query the database to fetch column $i+2$, and proceed to the next step. This approach guarantee that no AirPin data is fetch twice from the database without requiring to retrain in memory all AirPins of a city. To further improve the computation all AirPins in a column are sorted by increasing latitude. Revenues are computed in a top-down order and for each column a single index allow us to avoid reconsidering AirPins that are more than r meters Northern than the current spot of interest. This ensures that in each revenue computation only AirPins in a square or side $2r$ are considered (even though those at distance greater than r from the spot of interest provide no contribution to the revenue). In order to optimize database writing we maintain in memory an array of revenues for each user and write them all at the end of the computation of each city. Even those values that have to be written for each single AirPin are buffered in order to perform optimized queries that write several values at once.

In conclusion, the resulting computation is practically linear in the number of AirPins and does not perform redundant read/write database queries.

Bibliography

Martin Atzmueller and Juergen Mueller. Subgroup Analytics and Interactive Assessment on Ubiquitous Data. In *Proceedings of the International Workshop on Mining Ubiquitous and Social Environments (MUSE2013)*, Prague, Czech Republic, 2013.

Martin Atzmueller, Martin Becker, Stephan Doerfel, Mark Kibanov, Andreas Hotho, Björn-Elmar Macek, Folke Mitzlaff, Juergen Mueller, Christoph Scholz, and Gerd Stumme. Ubicon: Observing social and physical activities. In *Proc. 4th IEEE Intl. Conf. on Cyber, Physical and Social Computing (CPSCom 2012)*, 2012.

Martin Atzmueller, Martin Becker, Mark Kibanov, Christoph Scholz, Stephan Doerfel, Andreas Hotho, Bjoern-Elmar Macek, Folke Mitzlaff, Juergen Mueller, and Gerd Stumme. Ubicon and its applications for ubiquitous social computing. *New Review of Hypermedia and Multimedia*, 20 (1):53–77, 2014. doi: 10.1080/13614568.2013.873488. URL `http://www.tandfonline.com/doi/abs/10.1080/13614568.2013.873488`.

Martin Becker, Saverio Caminiti, Donato Fiorella, Louise Francis, Pietro Gravino, Mordechai (Muki) Haklay, Andreas Hotho, Vittorio Loreto, Juergen Mueller, Ferdinando Ricchiuti, Vito D. P. Servedio, Alina Sîrbu, and Francesca Tria. Awareness and learning in participatory noise sensing. *PLoS ONE*, 8(12):e81638, 12 2013a. doi: 10.1371/journal.pone.0081638.

Martin Becker, Juergen Mueller, Andreas Hotho, and Gerd Stumme. A generic platform for ubiquitous and subjective data. In *1st International Workshop on Pervasive Urban Crowdsensing Architecture and Applications, PUCAA 2013, Zurich, Switzerland – September 9, 2013. Proceedings*, pages 1175–1182, New York, NY, USA, 2013b. ACM. doi: 10.1145/2494091.2499776.

Saverio Caminiti, Claudio Cicali, Pietro Gravino, Vittorio Loreto, Vito DP Servedio, Alina Sirbu, and Francesca Tria. Xtribe: a web-based social computation platform. In *Cloud and Green Computing (CGC), 2013 Third International Conference on*, pages 397–403. IEEE, 2013.

Claudio Cicali, Francesca Tria, Vito DP Servedio, Pietro Gravino, Vittorio Loreto, Massimo Warglien, and Gabriele Paolacci. Experimental tribe: a general platform for web-gaming and social computation. In *Proceedings of the NIPS Workshop on Computational Social Science and the Wisdom of Crowds*, 2011.

Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Symposium on Visual Languages, Boulder, CO, USA – 3-6 September, 1996. Proceedings*, pages 336–343, New York, NY, USA, 1996. IEEE. doi: 10.1109/VL.1996.545307.