UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Master Thesis

Real time tracking and visualization of indoor social interactions

Candidate: Thomas Rouvinez[1]
Professor: Prof. Dr. Denis Lalanne[2]
March, 2015

University of Freiburg [3] // Freiburg
MSc. Computer Science • 1700 Freiburg • Switzerland

[1]thomas.rouvinez@unifr.ch
[2]denis.lalanne@unifr.ch
[3]University of Freiburg, www.unifr.ch

Acknowledgements

Thanks to:

- **Prof. Dr. Denis Lalanne** for offering me the opportunity to work in an interesting field of research, for his guidance and advice throughout this project.
- Special thanks to the team of the Swiss Integrative Center for Human Health (SICHH)[1], mainly **Dr. Jean-Marc Brunner**, **Dr. Jan Kerschgens** and **Dr. Mark Ambhl** for the opportunity to do large scale, live tests and bringing together the required hardware.
- Thanks to the CSEM[2], **Prof. Dr. Jean-Dominique Decotignie** for advice on RFID hardware and the modifications brought to the readers to support external antennas.
- My close family, my beloved Chiara and my friends for supporting me through this project and my brother Mathieu who helped me reviewing this paper.

[1] http://www.cish.ch/en, (March 2015)
[2] http://www.csem.ch/site/, (March 2015)

Management Summary

During this Master Thesis, we analysed the current state of research in the field of indoor tracking systems technologies. We studied their respective applications and issued a set of criteria that corresponds to the requirements of tracking social interactions. An indoor positioning system based on passive RFID tags carried by the participants of social events turned out to be the best approach. A complete indoor positioning system is then designed, including using RFID readers, capturing participants locations, manage and store the data, retrieve clean data, displaying this data in real time. A multi-tier system is created, combining RFID management and data capture, a web service to interface between the raw data and the final clients exploiting this data. The web service implements a set of methods to process the data, for both online and offline needs.

This project regroups the developments of multiple pieces of software: (1) a reader controller in Java, (2) a data management wizard, and (3) a live visualization. The reader controller manages the hardware and the tracking system. The wizard is used to manage the database, which is most useful to configure events. Finally, the live visualization has been created within a web page using Processing.js. The visualization is procedurally generated out of the data retrieved from the web service and aims at improving social interactions by proposing a live graphical representation of geographic interests within the room.

A live test has been carried out with 120 participants during a forum-like event. The data is analysed and presented with graphical representations such as stacked bar charts, stream graphs, etc. From observations performed during the live event, a set of hypotheses on the behaviour of the participants has been established. The system supported all these hypotheses and confirmed the observations. Further extrapolations based on the data collected are presented to highlight participants' behaviours that only the system can accurately track.

This Master Thesis has been achieved within a six-month time frame of combined research and implementation using agile methodologies (mainly Feature-Driven Development). Although work being performed by a single developer mitigates the value of agile methodologies, it allowed us to quickly iterate on the first transversal prototype.

Keywords: Visualization, Human-computer interfaces, RFID, Indoor tracking

Table of Content

List of Tables

List of Figures

List of listings

Acronyms

IPS	Indoor Positioning System
ITSI	Indoor Tracking of Social Interactions
RSSI	Received Signal Strength Indicator
CoO	Cell of Origin
TDoA	Time Difference of Arrival
AoA	Angle of Arrival
RTT	RoundTrip Time
RFID	Radio Frequency Identification
IDE	Integrated Development Environment
API	Application Programming Interface
MVCC	Multiversion Concurrency Control
CSRF	Application Cross-Site Request Forgery
WYSIWYG	What You See Is What You Get
FDD	Feature-Driven Development

1 Introduction

1.1 Motivations

Machines, calculators, super-calculators, computers, personal computers and now all sorts of smart portable devices retrace the evolution of human-conceived technologies. Although presenting ever larger design differences due to the research state and technological advancements, all these technologies share a common purpose: assist us where we reach our limits. Technological assistance used to be bound to a computer, but with the advent of smartphones and global technologies this is no longer the case. Therefore technological researches tend to bring assistance through any type of device, to any location. These researches are largely encouraged by the ever increasing expectations of the users toward technologies and a need for new ways to for optimizing one's free time. Nowadays expectations from our omni-present technology include guiding people through cities, stores, public transportation and even within their own homes. All these applications require systems analysing movements while actively searching and sorting data to provide relevant information and assistance. Although browsing and sorting data is a complex task, it has already been achieved efficiently. However, movements analysis imply that people's locations are known.

The GPS is the invention that made global positioning possible [9]. Due to the weakness of its signal, it only works in outdoor environments. Therefore there is a need for indoor positioning systems that can reliably take it from where GPS cannot be used reliably anymore. When designing an indoor positioning system, many technical challenges arise. Whether it is the way a user is detected, his/her position is established, data sampling and storage are dealt with, there are multiple options available. Giant companies such as Google or Nokia already propose indoor positioning systems based on Wi-Fi or Bluetooth multilateration using multiple access points and the smartphones of users to retrieve their position. Other systems may use optical devices, infrared, GSM-based tracking, etc. Creating a new indoor positioning system to track people has therefore already been achieved. Nevertheless, in this project we focus on using indoor positioning, not for the purpose of helping people find out where they are or where to go, but in order to optimize social interactions during events. In providing information the participants could not normally have during such events, we hope to help them identify more swiftly where people sharing their interests are.

1.2 Goals

In order to bring a valuable assistance and optimization of social events, this project aims at passively tracking participants by imposing as little constraints on them as possible. All data recorded should be stored in a comprehensive fashion, making it easier to retrieve and work on. From this data, we aim at creating a live visualization that brings new information to the participants, such as geographic distribution of interests within a room. Live data visualization requires a responsive system and only offers limited information. To get a more in-depth analysis of the events, this project implements a data crunching

system to work on data offline. Various types of statistics should be graphically illustrated in order to offer valuable insight and meaningful information that the human eye could not identify through regular observation processes. In an effort to make this system as extensible as possible, all raw data should be easy to extract for any required use. Therefore, the goals of this work are:

- Providing live visualizations of spatial interactions
- Offering visual reports of data after the event
- Offering access to the raw data collected throughout an event

1.3 Structure

Overall, this project has a two-fold approach: the first part focuses primarily on the creation of an indoor tracking system capable of tracking social interactions. The second part focuses on trying to identify ways to visualize interactions between participants during indoor events. Once an event is over, we will focus on offline data analysis to help discover social, geographic and social trends.

Social interactions visualizations are the main focus of this research which is to explore possibilities of visualizing social interactions. We focus on providing both a real-time representation of a social event and representations of the level of interactions with geographic distribution of interests.

Social interactions optimizations could be achieved by analysing the data of each participant and correlate them with other factors such as time, location, amounts of movement, etc. Various visualizations should then represent this information in an intuitive way. With such data we believe the quality of social interactions can be optimized.

An adaptive solution should be produced. Having a system that can easily be adapted, extended and reused is a top priority and will influence highly the choice of technologies. Although we focus on social interactions, the tracking devices should be reusable in any situation involving beacons to detect people or objects nearby.

In this work, we propose a new implementation of an indoor positioning system (IPS) we nicknamed Indoor Tracking of Social Interaction (ITSI). We aim at passively tracking the participants during social events or gatherings to offer both live and offline tools based on the data collected by ITSI. In chapter 2, state of the art is discussed and a list of possible technologies to build ITSI on is presented and are filtered through evaluation criteria. Chapter 3 proposes a general architecture of the system to bind all essential development parts together. Chapter 4 details the selection process and software management of the hardware. In chapter 5, technologies for the middleware are evaluated and the complete specifications for its roles are given. The final part of the system is described in chapter 6 with client-side applications. We provide an example application of a live representation of the data ITSI captures. Finally chapter 7 presents results of a large-scale test of ITSI.

2 State of the Art

In this section we present the current state of research (1) in indoor positioning systems in chapter 2.1, (2) RFID technologies in chapter 4, (3) technologies for social events in chapter 2.3 and (4) GUI prototypes for our system in chapter 6.

2.1 Indoor positioning systems

Finding one's position in the world without actual use of a map and compass was made possible through GPS technologies. With accuracy errors ranging from 3 to 10 meters, it is efficient enough to be trusted to point directions while driving or guide us through cities. The GPS uses radio waves picked up by at least three satellites to determine the location of the device through multilateration. It then sends back the information to Earth, also with radio waves, at a frequency of 1575.42 MHz[3]. These high-frequency signals do not respond well to interferences and thick construction materials. This results in large inconsistencies and errors in positioning as soon as the direct line of sight between emitter and receiver is broken. Therefore the GPS cannot reliably be used for indoor positioning. This leads to the development of new technologies of IPSs for use beyond the availability limits of the GPS service. IPSs are based on a plethora of technologies or combination of technologies to achieve their purpose.

In the following chapters we present the criteria of evaluation for IPSs as presented in [10] and a non-exhausive list of the most relevant IPSs in their current state of development. We proceed by types of technology used and present their respective applications. Indoor positioning is still a relatively new topic of research and most of the products introduced here are either experimental or trying to push the technology to create a new market.

2.1.1 IPS evaluation criteria

Previous work has been done in the field of IPSs evaluations. We present in figure 1 the IPS selection criteria as found in [10], page 15. We cover less technologies as in [10] but focus on our specific application's requirements to identify the best candidates among the IPS-powering technologies.

With the above-defined user requirements for our specific application of indoor positioning, we may proceed and analyse the different available IPSs by technology types. To evaluate these IPS we will use the performance parameters and weight them against the user requirements of this project. To make sure each performance parameter is as unambiguous as possible, we define them exhaustively according to [10]:

[3]Source: http://science.opposingviews.com/doesnt-gps-work-inside-building-18659.html, (October 2014)

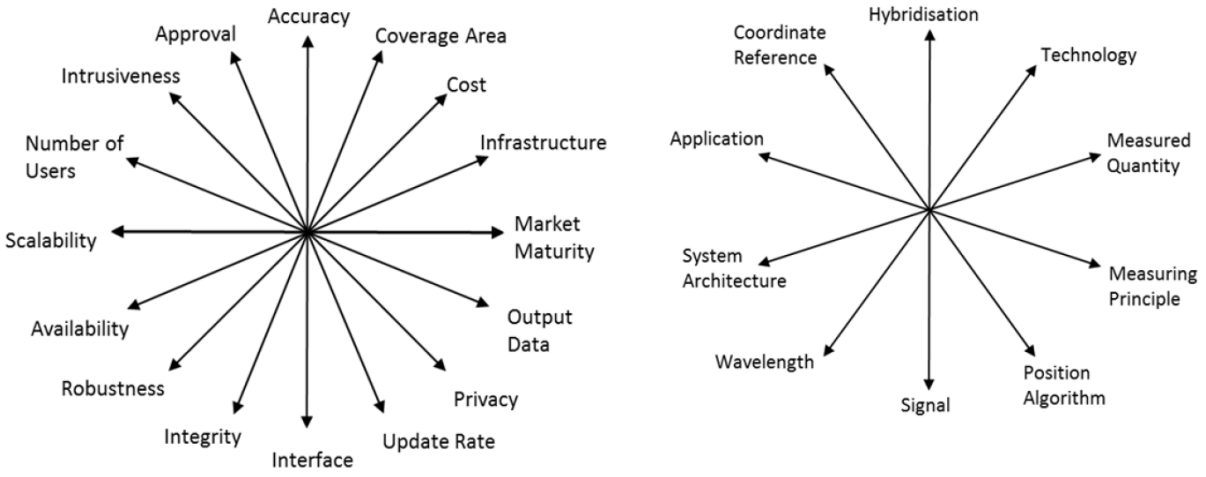


Figure 1: IPS performance parameters (left) and user requirements (right).

Accuracy in IPS can be described as the variation of measurement between an X' - Y' coordinates pair at a given time compared to the real X - Y values at the same given time. Accuracy is an essential criteria for IPS and should, depending on the application requirements, impact a great deal the choice of technology.

Coverage determines the spatial area in which the IPS must be able to perform with the desired level of accuracy. According to [10] an IPS should be categorized in one of the following categories:

- Local coverage: a non extendible, small and determined coverage area.
- Scalable coverage: same as a local coverage but featuring the ability to scale up or down the covered area by adding/removing hardware.
- Global coverage: coverage that applies to worldwide applications. Example: GNSS[4] and celestial navigation.

Cost regroups all the expenses required by an IPS for installation, per user device cost, pet room costs and maintenance costs all together.

Infrastructure regroups the types of technological implementations required for the IPS to work correctly. Such implementations can be markers, tags, detection beacons, reusing parts of the current indoor setups or not, etc.

Market Maturity describes the state of development of an IPS. These states are usually (1) "prototype" where the product is being designed, (2) "in test" when a prototype is mature enough to run real tests on it and (3) "market-ready" product. The level of market maturity is a very influential criteria to estimate the time of production needed to get one's product ready. It also influences the development costs which are correlated to the setup costs.

Output data represents the type of data returned by the IPS. Such data can feature 2D or 3D coordinates for positioning, a relative/absolute position and data derived from these such as position prediction (Kalman and/or particle filters for example), target speed, etc.

Privacy As previously explained in the user requirements of our specific project, privacy is influenced by the type of equipment the users require. Whether the technology depends on the user's smartphone, active or passive tags for detection, or no technology at all influences the level of acceptance of the IPS. This criteria is tightly coupled with the Intrusiveness criteria.

Update Rate determines the frequency at which the technology can register an X - Y pair of coordinates. The frequency can be of three types: (1) periodic at a given regular frequency, (2) on request by the user or (3) on event based on sensor acquisition or data received.

[4]GNSS: global navigation satellite system.

Interface describes all the possible ways to connect to an IPS. Such connections are usually achieved through RS232, Ethernet RJ45, USB, Wi-Fi, Bluetooth, etc.

System integrity is the ability of an IPS to detect when a part of itself is not working correctly anymore. It spans across systemic data losses, parts of the system going down or the whole system not working anymore. The integrity criteria is coupled with the robustness and availability criteria of an IPS.

Robustness represents the endurance of an IPS. With various hardware dispatched to cover full areas, hazards such as physical damage, theft, wireless signal jamming, spoofing, or man in the middle accesses can happen and seriously alter the global performances of the system.

Availability quantifies the likelihood an outage might happen and how long it would last. IPS can implement various mechanisms to mitigate interruptions of service such as automatic detection of failures and alert systems.

Number of users is the quantified limit of simultaneous users who can use the system. This value is often a compromise between the number of users detected and the refresh rate of the system.

Intrusiveness relates to the level of user disturbance imposed by the IPS. Some systems will require the users to wear a certain amount of technology or to allow the use of their private smartphones with access to data.

Approval ascertains the certifications and authorizations required to use such a technology. There may be regulations in countries forbidding the use of IPS technologies.

According to [10], the performance parameters have to match the user requirements in order to determine the right technology for the right application. Identifying the best technology for the purpose of our project highly depends on the following requirements:

- **Privacy management** is one of the most important expectancy of the technology. Since we focus throughout the development on assisting social events, respecting the privacy and anonymity of the participants is a key factor in technology acceptance. This implies that no personal data or any access to it is allowed to avoid spooking the participants from using our system.
- **Passive technology** should be favoured over requiring the participants to use attached or personal devices. Such devices range from smartphones to laptops or any portable technology equipped with networking capabilities. The main reason behind this logic lies in the fact that participants use their own equipment for their own purpose and rarely agree to install applications on their smartphones or to end up with a depleted battery by the end of the event.
- **Zoning over tracking** is a difficult choice to make. With tracking it becomes possible to extract a lot more information than with zoning, but at the cost of participants' privacy. With zoning the data gathered is less accurate as we cannot say precisely within a perimeter where a participant is unless that person is picked up by multiple zones. In this case multilateration helps to more accurately render the position of the detected person.
- **System portability** is another requirement that may prove useful but would not be mandatory. On the one side, a system strongly coupled to a given indoor environment will improve the reliability of the system while making its setup easier (calibration is performed once). On the other side, having a portable solution allows for reconfiguration and adaptation to new environments while keeping the costs low.
- **Installation costs** are the last requirement to consider. Indeed it can be worthwhile to increase the costs of the IPS, provided that it grants real advantages. In the scope of this project we experiment and will select an affordable solution but consider all available alternatives.

We may now analyse a selected subset of technologies in the light of the aforementioned and described criteria. Note that we may not always consider all the criteria as some technologies are not documented

enough due to their prototyping market maturity. Each IPS technology is analysed as being the only one used — no fusion allowed.

2.1.2 Cameras

Cameras-based IPS use three different techniques to track visual elements: (1) Sequential images analysis, (2) reference points tracking and (3) tracking without any reference. In the first case multiple cameras are required to take a sequence of pictures in all the covered areas. By using time stamps of the pictures one can retrace the movements of a target. This solution presents issues for real-time performances as image processing combined with geographical target representation of the target is computationally expensive. On the positive side, this technique allows the tracking of multiple targets at the same time and gather more information via computer vision algorithms. In the second case, the task is simplified by using visual tags. These tags are usually a pattern of coloured dots or visual codes that provide useful information such as scale and orientation at the same time as the X - Y tracking. Such systems go from a few centimeters to sub-centimeter accuracy readings for professional solutions. Although this solution may not be expensive to set up, it still requires additional hardware and load-ready processing capabilities. Finally the third case requires no tag to track objects but this approach is used mainly to follow objects that stay in a predefined field of view.

Using camera-based IPS presents many advantages and is one of the best solutions for indoor tracking. New systems of surveillance are exploiting multiple sets of stereoscopic cameras combined with infrared-based depth sensors. The information gathered combines local position, orientation, scale, and more can be obtained via computer vision techniques. The richness of information gathered is however impaired by the requirements of camera-based indoor tracking. Indeed, all the approaches using cameras have one negative side in common: the hardware is set in stone. Indeed this IPS requires all the cameras to be mounted in determined locations and calibrated. In absence of this prerequisite, having a self-configuring system is hard to achieve and may result in tracking drift and inaccuracies.

Another clear downside of this technology is privacy. For a camera-based tracking system to work accurately, it requires high quality pictures. Since these pictures are used to detect small patterns or notice changes at least at the centimetre level, the user can assume he is being monitored. Hence the users of video IPS often associated it to surveillance rather than a tracking system.

Table 1: Evaluation grid of Camera-based IPS.

Criteria	**Appreciation**
Accuracy	Centimetre to sub-centimetre accuracy
Coverage	Scalable
Cost	Cameras are affordable, expensive dependencies
Infrastructure	Needs to be mounted and calibrated
Market Maturity	All types of market maturity available
Output data	2D, 3D coordinates, scale, orientation
Privacy	Close to none depending on camera resolution
Update rate	Configurable, depending on processing power
Interface	Wireless, RJ45, USB, HDMI
System integrity	Easy to detect malfunction (based on camera feed)
Robustness	Cameras can be mounted in unreachable locations
Availability	Rare hardware failures, high availability (99.9%)
Number or users	Software dependent, medium to high
Intrusiveness	Limited with references
Approval	Approved in USA and EU

Overall the camera-based IPS solution provides a rich set of information. Moreover it packs a level of accuracy no other technology can significantly surpass. However this IPS requires a static and calibrated setup in order to achieve the desired levels of accuracy and coverage. Hence the difficulty to modify, adapt or extend the setup without reconsidering the whole installation. Although the price of the cameras

themselves is not as expensive as other technologies, the dependencies needed in terms of processing power, data storage and network bandwidth easily make up for it. Therefore camera-based IPS tend to be expensive in the end. Finally, the privacy is the most important setback as for the use we would have for this technology within this project.

2.1.3 Infrared

Another technology working with optics is infrared. It works on wavelengths that the human eye is unable to perceive. According to [10], the range of infrared technologies is large but there exists three popular ways to use it: (1) Through active beacons, (2) infrared imaging from thermal radiations, or (3) artificial lights. In the first case, infrared receivers are placed in the indoor environment with a referenced location. The target the system must track wears an active infrared emitter that pulses unique codes at a given rate. Once a pulse is detected on one of the receivers, it can be identified and the Angle of Arrival (AoA) computed. Therefore a location can be determined. The accuracy of detection can be improved by adding more receivers to the environment. The second case features passive detection based on the long wavelengths of the infrared spectrum. These wavelengths are used for thermal detection. By identifying the temperature of humans, one can track someone without requiring the target to wear any identification tag. Tracking is then performed by looking for the same temperature through multiple sensors and apply triangulation. Finally, the third case makes use of artificial light projection to capture a scene. Microsoft's Kinect can, for example, establish a 3D representation of depth using projected infrared dots in the scene.

Out of these three solutions, the first one has to be excluded from the list of potential technologies we may use for this project. Indeed, the high number of infrared receivers and computation of triangulation do not fit real-time applications. The second solution based on thermal detection must also be discarded for its instability. Indeed, in environments with multiple heat signatures or sunlight influence, the results will become very unstable. Therefore only the third option can be considered with the use of Kinect[13]. Packed with a complete and well documented SDK, the Kinect offers opportunities to harness stereoscopic vision of a scene. It is then the best infrared-based solution in the context of this project.

Table 2: Evaluation grid of Infrared-based IPS.

Criteria	**Appreciation**
Accuracy	2cm at 2m, range up to 3.5m
Coverage	Scalable
Cost	200$ per unit, setup is expensive, need third party softwares
Infrastructure	Same as cameras: mounts and communication infrastructures
Market Maturity	Market ready
Output data	Depth and camera feed to provide 2D, 3D, Kinematics
Privacy	Cameras are low res, but still cameras
Update rate	Depth and RGB cameras: 640x480px at 30Hz
Interface	USB
System integrity	Easy to detect malfunction (based on camera feed)
Robustness	Cameras can be mounted in less reachable locations, maximum 3.5m
Availability	High availability (90%)
Number or users	Software dependent, built for up to 4 XBox players
Intrusiveness	None
Approval	Approved in USA and EU

We have seen that infrared-based technologies can be used to track users in indoor environments. But the technology doesn't lend itself well for the purpose on its own due to external factors that tamper with the results. The gaming industry has developed the Kinect that uses an infrared camera to create a depth map and track objects. It harnesses the accuracy granted by fusion of both the infrared and RGB cameras combined with an array or microphones. For the usage required in this project, a Kinect-based system remains expensive and lacks adaptability as the cameras have to be mounted in a static setup.

2.1.4 Wi-Fi Positioning System

Indoor tracking requires additional hardware in any case. With most current buildings providing Wi-Fi coverage, it makes economical sense to reuse the already existing installations. Moreover the range of Wi-Fi and its ability to emit through walls make it a great indoor solution. Common Wi-Fi receivers and emitters feature RSSI readings. It is therefore the main measurement for Wi-Fi tracking. RSSI is the value describing the estimated strength of the signal received. Various functions taking the RSSI as a parameter can then help determine the distance between the emitter and the receiver. There exist four main types of techniques to perform RSSI tracking with Wi-Fi according to [10]:

- **Propagation modelling** consists in describing a model that simulates how the Wi-Fi signals are propagated through the environment. Indeed, the RSSI may not always indicate the direct route between the emitter and the receiver, but the strongest one. Wi-Fi uses radio signals which are altered by the surrounding environment. Indeed the transmission speed in an empty environment such as space reaches the speed of light. However if a wall, glass, door, etc. is in the way, it will slow down the signal. Therefore the RSSI is environment dependent. In an attempt to correct the RSSI values, models of propagation can help reduce the amount of calibration required.
- **Cell of Origin** is using Wi-Fi emitters as beacons. The beacon receiving the strongest signal from the tracked device is assumed to be the location of the device. Although simple, the accuracy is of roughly 50 meters.
- **Fingerprinting** is a way of adjusting the RSSI to the environment. A "radio map" has to be created to act as a ground truth for the RSSI. Various RSSI values are recorded in the environment during a calibration phase, then the values are taken are ground truth for positioning. The RSSI are then used to compute the Euclidian distances and the smallest one is taken as the real position. The downside of this approach is the required calibration time every time the environment changes. Also the RSSI can vary from one Wi-Fi chip to another and depends on the orientation of the device. On the upside the environment is taken into account in the distance computations, hence increasing the accuracy.
- **Multilateration** with Wi-Fi signals can be based on RSSI, Time Difference of Arrival (TDoA), Angle of Arrival (AoA) or Round Trip Time (RTT).

Fingerprinting based on RSSI is currently the best known solution to achieve indoor positioning with Wi-Fi. The fact that modern Wi-Fi hardware can by default compute the RSSI reduces the expenses of such a technology. Therefore for this project we consider fingerprinting on RSSI as a potential technology.

Table 3: Evaluation grid of Wi-Fi-based IPS.

Criteria	**Appreciation**
Accuracy	between 2m and 50m
Coverage	Scallable
Cost	Affordable hardware, scale dependent
Infrastructure	Wi-Fi access points, tracked devices on users
Market Maturity	Prototype / In test
Output data	2D coordinates and signal strength
Privacy	Requires access to personal smartphone
Update rate	30hz for cameras, need processing
Interface	Wireless, RJ45
System integrity	Scale dependent, down link can be identified
Robustness	Out of users' reach, very good
Availability	High availability (99.9%)
Number or users	High
Intrusiveness	Requires access to personal smartphone
Approval	Approved in USA and EU

We conclude from the Wi-Fi approach that the technology is affordable and reaches good levels of accuracy. Most of the hardware can be reused from currently existing Wi-Fi installation within buildings, thus easing the deployment of a solution based on this technology. There exists various techniques to perform Wi-Fi indoor localization and companies such as Google and Nokia are already testing their own implementations. Google for example released an experimental version of their "Maps" application that alternates between GPS tracking outdoor and Wi-Fi IPS to rely on when the GPS is not accurate enough.

In the context of our project, the accuracy and ease of deployment are clear advantages. However this system works with a device the user must carry, preferably a smartphone. In this case not only the privacy of the users is impacted, but the Wi-Fi will drain the batteries of their phones rapidly. Moreover with fingerprinting the calibration required is time-consuming and needs to be performed regularly to ensure good accuracy of the system.

2.1.5 Bluetooth

Bluetooth technology relies on radio frequencies to communicate with other devices. Bluetooth devices can be categorized in three classes: (1) class one with up to 100 meters range, (2) class two with up to 10 meters range and (3) class three with up to 5 meters of range. The working principle behind Bluetooth indoor positioning is relatively similar to Wi-Fi IPSs. Indeed it is based on placing beacons at key locations and create a grid of detection. Upon stepping into a zone, a user will be detected and the location of the beacon taken as assumed user location. If a user is picked up by multiple beacons, then the accuracy increases as the user has to be in a smaller area covered by all the concerned beacons. In addition, the RSSI is also available with Bluetooth so that multilateration may be performed.

An example of an inexpensive Bluetooth IPS is described in [2] where cheap Bluetooth receivers are created from standard technology and aluminium foil. The results of this experiment show that a 2 to 3 meters radius of detection grants a good enough detection accuracy for indoor positioning.

Table 4: Evaluation grid of Bluetooth-based IPS.

Criteria	Appreciation
Accuracy	Class-dependent
Coverage	Scalable
Cost	Cheap
Infrastructure	Beacons and one device to be tracked
Market Maturity	Market ready
Output data	Triggered receiver identifier, RSSI
Privacy	Good, has to be in a detection zone
Update rate	Long identification time, fast readings
Interface	USB, Wireless
System integrity	Scale-dependent, can be hard to identify a down beacon
Robustness	Beacons are accessible
Availability	Scale-dependent
Number or users	Large
Intrusiveness	Requires a device on the user, smartphone preferably
Approval	Approved in USA and EU

Bluetooth and Wi-Fi are polyvalent with regard to the approaches they support for user detection. Compared to IR and RGB cameras where tracking is deduced from the processing of the data, Bluetooth and Wi-Fi may be used either for tracking or zoning. Zoning implies that users are detected only within a given zone and that no X - Y coordinates are extracted. Although significantly less accurate, it opens possibilities to better fit specific needs via a scalable infrastructure. It also makes data processing much lighter, which reduces the processing power and amounts of data storage required. Nevertheless, even with zoning in Bluetooth the user's phone has to be used and this is not optimal considering our user requirements.

2.1.6 Sound

Electromagnetic waves are not the only solution to propagate data. Indeed mechanical waves such as sound also work. The difference is the media that carries information through the air. Apart for that the principle remains the same: distance measurements are combined with multilateration to determine indoor location of an emitter. Three or more receivers with known X - Y coordinates should pick up an ultrasound signal. Then to avoid issues with synchronization, the TDoA is used. Two types of signals are transmitted, usually ultrasound and radio waves and the difference of time of arrival at the receiver is computed. This allows for the accurate computation of the distance as the difference of transmission speed between ultrasound and radio waves is known.

There exists mainly two types of sounds used for IPS: (1) ultrasound and (2) audible sound. Ultrasounds are the common approach as the users cannot notice it, we will hence base our analysis on ultrasound IPS. In the ultrasound realm the tracked objects can be attached either to a passive or to an active system. The active system will broadcast ultrasounds to be picked up by multiple receivers. This mode of operation is prone to overlapping as all the targets need to emit signals in a well defined time slot. Therefore scalability is directly affected as well as the update rate of the measurements. Indeed with an increasing number of targets to track comes the need for more time slots. Thus the total time required to scan all the targets is increased. Another issue is the energy consumption of wireless active ultrasound emitters which requires power schemes to prevent draining the batteries too quickly.

The passive system flips roles by having active emitters as beacons and the tracked target only wearing a passive device. The main advantage lies in the passive mobile device never having to transmit any data. Therefore the user's privacy is ensured. The scalability issue with passive systems disappears as the mobile devices do not need to transmit any data anymore. Thus the system is easily scalable without risks of overlapping signals.

Table 5: Evaluation grid of Sound-based IPS.

Criteria	Appreciation
Accuracy	1 to 5 cm within 10 meters range
Coverage	Scalable
Cost	Cheap devices, common hardware
Infrastructure	Set of active beacons & mobile devices
Market Maturity	Protoype & market ready
Output data	2D coordinates
Privacy	Excellent
Update rate	1Hz is achievable
Interface	Installation dependent
System integrity	Failure detection is hard, scalability-dependent
Robustness	The active beacons have to be protected
Availability	High availability
Number or users	Scalable
Intrusiveness	Close to none
Approval	Approved for USA and EU

The passive ultrasound IPS technology we focus on uses the same principles as Wi-Fi or Bluetooth-based IPS. The ultrasounds only fit indoor usage as they are highly sensitive to temperature changes that outdoors environments feature. The 10 meters maximum range of ultrasound IPS is also a limitation for outdoor uses but an advantage for indoor tracking. Indeed, with small radius of detection created by each active beacon, the infrastructure can be modulated and organized to best fit the environment. The technology's scalability is only depending on the number of active beacons installed. From the user point of view, no use of personal smart phones, data plan or battery is required. This first part enforces privacy. The fact that the mobile device does not need to send data for actual tracking to happen makes the user even more transparent to the system.

2.1.7 RFID

Radio Frequency Identification (RFID) is achieved through RFID readers and tags. An RFID reader is the combination of a set of antennas (directional or omnidirectional) and an emitter. Radio frequencies are continuously transmitted and any tag in the field of detection will be triggered. Tags contain information they transmit upon activation. This data usually consists of a unique identifier and, depending on the type of tag, additional customizable data. Proximity detection is also known as Cell of Origin (CoO). With CoO the only information obtained indicates whether a tag is in the detection radius of an antenna or not.

Tags come in two types: (1) active and (2) passive tags. An active tag is powered by a battery. Upon receiving radio frequencies the tag activates and uses it's own power to emit its data. The passive tags have larger antennas to collect the energy of the radio frequencies and reuse this energy to send a signal. Therefore passive tags have no need for dedicated batteries. As the energy from the incoming radio frequencies is used to power the tag, passive tags have smaller detection ranges than active tags. The current ranges go up to 16 meters for passive tags, respectively 100 meters for active tags. Another difference among tags is the ability to write data on them. Most tags are read-only but since RFID technology is widely used to trace shipment information, tags must allow writing operations as well.

RFID IPS consists of a set of readers positioned in indoor environments. If a tag is read in one zone of detection, the tag is assumed to be on a circle or line (depending on the type of antenna) at a given distance computed from the RSSI [6]. If a tag is read by multiple readers at the same time, then the region in which the tag is detected becomes more accurate. Therefore accuracy of RFID IPS is dependent on the number of antennas and their read range. Note that RFID readers may come with adjustable read power. This allows for the creation of combinations of read ranges. It is also possible to perform multilateration based on the RSSI and reading a single tag from three different readers.

Table 6: Evaluation grid of RFID-based IPS.

Criteria	Appreciation
Accuracy	Range and Tags type dependent
Coverage	Scalable
Cost	Range dependent, between 500-2000$ per antenna
Infrastructure	Antennas and tags
Market Maturity	Market ready
Output data	Tag ID, RSSI
Privacy	No personal information shared
Update rate	Up to 10Hz per tag
Interface	USB, RS232, RJ45, Wireless
System integrity	Failure detection is hard, scale dependent
Robustness	The RFID reader has to be protected
Availability	High availability
Number or users	Scalable, reader-dependent
Intrusiveness	Very limited
Approval	Approved for USA and EU

RFID-based IPS use the principle of choke point: track tags only at choke points and therefore don't focus on tracking, but zoning. The adjustable read ranges and scalability of the system allow to create various configurations that can be adapted to various environment. The privacy of the users is also kept intact as they are not always tracked and no personal information can be found but the tag identifier. RFID technology is preferably aimed at industrial applications but could easily fit other purposes.

2.2 Related IPS solutions

IPS with the intent to enhance social interactions have already been developed: McCarthy [11] proposed a system of interactive screens offering information about participants and one to display its interests when near a screen are developed. Compared to our solution, both require the participants to be close to a

screen and they do not track movements.

Konomi [8] proposes a solution where RFID technology is used in the same way ITSI does, but only close to screens and proposes force directed graph-based visualizations to either passively review social interactions, or actively navigate in these graphs.

The "Conferator" [1] system offers the same functionalities as ITSI and also relies on RFID technologies, but their accuracy is at a room level. This mean s that a participant will be assigned to a room, because this system is supposed to monitor multiple rooms. ITSI focuses on direct social interaction — at a finer resolution, since in the current evaluation detected zones are around small tables, and even finer resolution could be even considered since they depend on the reading range of the chosen RFID readers.

The "IntelliBadge" system is presented in [3]. It is an accademic experiment based on RFID active tags and aims at providing live tracking and visualization for large events (5000 participants). The participants were tracked only at kiosks with large displays to see the information. It aims at bigger events, and mostly only provide more information about participants rather than information about social interactions.

In [4], RFID tags are used to provide services such as "contact information exchange, people and publication comparison, learning new relevant things, etc". Though this system uses the same technologies as ITSI, it is clearly service-oriented rather than social interactions-oriented.

Although all these systems are close from a technological point of view, they put the RFID readers either on walls or at booths during their respective events. ITSI places the readers directly at the center of the discussions. Another major difference is that all these other systems focus on larger events and often have to dilute the number of RFID readers per room to cover more ground. Finally, most of these systems use big screens as a mean for the participants to log in and consult data captured from their respective IPS, whereas ITSI displays for all participants meaningful and useful information about the social interactions. The aforementioned systems we reviewed are focused mostly on data exchange. ITSI, however, focuses on supporting data capture and analysis to help with social interaction. The other systems bring useful tools tracking participants only in precise locations, mainly well placed booths or screens in events. Note that the research on non verbal analysis of social interactions is also a furnished topic of research, as Gatica [5] explains. ITSI could be used as a basis for such systems but would need to be used in fusion with other technologies as ITSI's main focus is about verbal interactions.

2.3 Technologies for social events

In the previous chapters we have explored the most popular ways to perform tracking indoors. We have compared these technologies with the user requirements of this project and determined key factors to choose the correct technology:

- The user shouldn't be asked to use his/her own equipment to contribute to the tracking. There are two main reasons: first the user's equipment may contain personal data. Even if the application on the equipment does not access personal data, the user will be reluctant to authorize the use of personal equipment such as smartphones. Second, during a social event it is likely the users will need to use their equipment to store data, browse and exchange information. Most of the tracking technologies require networking and processing capabilities which will drain the batteries. Hence the user's access to his/her own equipment will be limited.
- Globally zoning should be favoured over tracking. As pictured in figure 8, the system should be able to detect users around key locations. There is no need to know precisely where a user stands as we are interesting to know who is near enough to other users and base our analysis upon this. Moreover zoning requires less processing power and produces less data to store. Therefore it is more efficient in the context of this project. Another key aspect of zoning is psychological: the users know they won't be tracked throughout the whole event, but only at key locations. This difference helps the user accept the technology as a tool rather than surveillance.

Considering the first key factor, all technologies using the users' smartphones should be discarded. This applies to Wi-Fi and Bluetooth technologies. From the second key factor, constantly tracking IPS should be discarded. This applies to cameras and infrared cameras. Finally, although the sound-based IPSs regroup all the requirement, their instability discards them for our usage. Therefore we deduce that RFID solutions are the fittest technologies to build ITSI on.

2.4 RFID Technology

The RFID technology finds its roots in the research of radio-based technologies. While at first such researches were only oriented toward the transmission of data from an emitter to a receiver, other needs appeared. As soon as identification through the use of radio waves has been achieved, the RFID was born. Although not the purpose expected from radio frequencies, radio identification of objects was first experimented on by Sir Robert Watson-Watt [5]. He is mainly known for his invention to detect aircrafts in the sky: the radar [12]. A radar is both a transmitter and a receiver that broadcasts a radio signal to be reflected by aircrafts. This reflection is perceived by the receiver. Therefore an aircraft can be located within a radio broadcast range. Yet this technology can not be called RFID since the objects detected are not identified. The purpose of having a radar was therefore still unachieved. Indeed being able to identify the aircraft detected as either a friend or a foe was of capital importance. Sir Robert Watson-Watt developed the Identification Friend or Foe (IFF) which is a transponder aircrafts carry. When a radar radio signal is sent and hits an aircraft, the transponder returns the signal with an identification code. This is the first RFID system conceived[6].

Later developments focused on more affordable commercial uses of the RFID technology. Among the first commercial applications was the creation of security tags that would trigger security if their identification was not correct when read at an exit of shops, buildings, etc. Current applications for RFID technology englobes [6]:

- Store product identification: delivers product information and anti-theft security.
- Parts identification: tags store information about a product or a part of it. Identifying an object and storing data on its RFID tag allows also to ease production control.
- Personnel access management: this is used as an automation of work control and privileges - security enforcement.
- Supply chain control.
- Asset tracking: creates a trace of all control points a product went by. This also applies to airline luggage identification and routing automation.
- Vehicle tracking and automatic toll payment.
- Livestock tracking: small RFID tags can be implanted in animals or humans for tracking.

Lately the RFID technology has been used for more than tracking stocks or packing information on a product. In IPSs it is commonly used in the same way as Bluetooth: create landmarks. These landmarks allow to correct the drift that another IPS could have accumulated over time. Greater accuracy and position correction are achieved although fusion of tracking technologies.

2.5 Conclusions

There are many technologies already available that can be turned into the foundation of an IPS. Among these we discarded all that require the users' smartphone to avoid energy draining and all that intrude too much on users' privacy. We described the notions of tracking, zoning and coverage with regard to making observations only in key locations (landmarks). This research points us toward using RFID technology to detect users near landmarks. The users have to wear a passive tag and are positioned only when within read range of a landmark. Even though this only constitutes a selective positioning system, it satisfies the requirements of this project and can always be updated in the future to work in combination with

[5] http://en.wikipedia.org/wiki/Robert_Watson-Watt, (January 2014)

[6] Source: http://www.radio-electronics.com/info/wireless/radio-frequency-identification-rfid/development-history.php, (January 2015)

other technologies through early or late fusion. Early fusion merges the raw data together while the latter merges processed and cleaned data to validate the results of the sensors. RFID technology has a long history of applications and developments, it is market ready and available.

3 General Architecture

3.1 Overview

The overall design is split in three tiers according to figure 2, from left to right: First we explain how participants are tracked and how their data is gathered in chapter 3.2. Then the middleware that processes and stores the data is explained in chapter 3.3. Finally the client-side sample applications we propose through this project are presented in chapter 3.4.

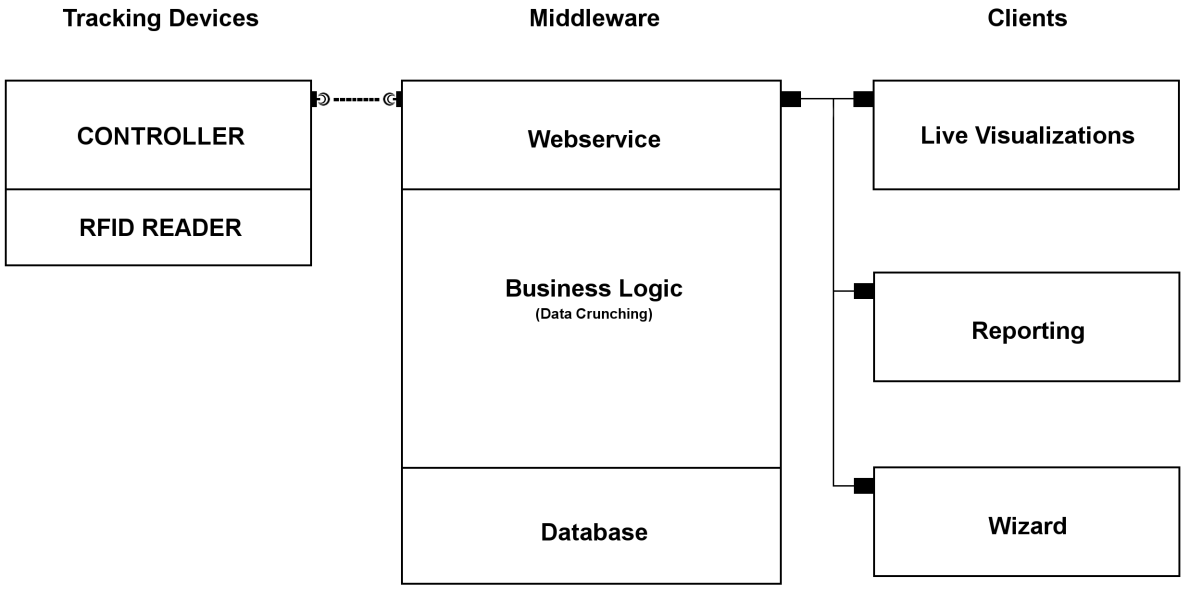


Figure 2: Overview of the system.

In this section we focus on providing an overview of the high-level architecture of the system. The system is based on beacons with a limited detection radius. Each of these beacons is made of a single RFID reader paired with a controller to command reading and forward the results to a server which will store and process the data. The server also offers a web service for ease of data access.

3.2 Tracking devices

The first tier of the solution's architecture consists of both (1) an RFID reader and (2) a controller to power the reader and to provide access to the data. As explained in 2.3 we perform zoning rather than tracking and for this purpose we need beacons that detect people and / or objects within a known radius. To achieve that, we use a set of beacons placed in an indoor environment. Each detection zone is defined and assigned a unique zone identifier. As soon as a tag enters a zone, the tag identifier, a time stamp as well as an RSSI value are recorded by the RFID reader. By relating a tag identifier to a zone we are able to locate a participant or object within the detection zone of the beacon that picked up the tag.

Once the RFID reader has picked up information, the data is transferred to the controller. The concept of the controller is to be able to pick up the reader's data, apply light pre-processing and forward the information to a server. The medium of data transmission depends on how the beacons have been conceived. Two conceptual implementations for the beacons are proposed:

1. Tables can host an RFID reader and a controller under the table plate to pick up any tag nearby. The main advantage is that a tag detected with a strong RSSI indicates reliably that a person is active in that zone. They also provide a natural environment to support discussions.

2. Lamps are another approach that can be less intrusive and offer more possibilities to give feedback through control of the light's colour. However, determining if a participant picked up in this zone of detection is really active and not just standing there is less obvious.

In any case, detecting a tag in a zone does not provide sufficient evidence that a social interaction is happening. Other factors such as the time spent in a zone or variations of RSSI need to be considered to enhance the accuracy of the system. More accurate indoor positioning can be achieved by ensuring detection zones are overlapping. Provided at least three RFID readers pick up a tag at the same time, multilateration is applicable. The way the beacons are placed in the environment is fully dependant on the type of application and data we are interested in. For example during forums, the centers of attention are the tables, therefore the beacons should be placed on each table to record who spends time at which table over the duration of the event. However, if we are more interested in establishing a trace of a participant's whereabouts, having multiple long range beacons to harness multilateration is better suited.

3.3 Middleware

The second tier of the solution's architecture acts as a middleware. On one side, it should be able to receive raw inputs from the beacons and store them in a comprehensible way. On the other side, the data stored should be remotely retrievable from clients who offer purpose-oriented information abstracted from the raw inputs. To achieve this, the middleware is split in three parts:

1. **A Web service** layer is the actual implementation of the interfaces. It handles all communications between the readers, the database and the client's applications. Indeed, thanks to functions storing data in the database, the web service picks up the data from the RFID readers and saves them. Through a set of selected functions the clients are able to extract the data required to perform visualizations or reporting. Data extraction to feed these functions is performed in the business logic layer. The web-service should ensure secure data transfer through encryption and also provide privacy for the participants, as no names will be stored in the same database — only the identifiers of RFID tags.

2. **A business logic** layer to handle data extraction. This process includes (1) data preprocessing for clear data storage, (2) data crunching to provide higher level information and (3) data optimization to avoid storing and sharing useless data. The business layer is acting as an interface between the web-service and the database to perform secure and fast queries on the data.

3. **A database** is the heart of the system as it will store all the information from the RFID readers. The database structure should remain simple for reasons of performance and ease of access.

The middleware is the only tier of the system that is completely generic. Indeed the beacons may change from a hardware point of view but are dependent on the RFID technology. Hence the information delivered will always be the same. The same applies to the client applications that will request data from the middleware to process it further. Therefore the middleware is a key point to interface all the elements of the system.

3.4 Client applications

Finally the last tier of this project is composed of various client-side applications. In the scope of this project, we focus almost exclusively on the live visualization. An example of a concept is presented in Figure 8. Given the live nature of these applications, data exchange and representation have to find the best trade-off between refresh rate and data renewal.

Other types of client applications focus on offline data and on offering easy access, or focus on configuration of the system to further process input data. In the first case, a wizard will be required to provide ease of access to the database via CRUDs[7]. In the second case it will be made sure the web-service offers enough flexibility for other types of applications to harness the existing system.

[7]CRUD: set of creation, read, update and deletion operations

4 RFID-based tracking system

In this section, we focus on the first tier of the system: the physical indoor detection and localisation of people. We describe the history of the RFID technology, its first applications and the evolution it followed. We also present our choices of hardware for this project — for readers, antennas, and tags. Finally, we describe the way the system handles data collection and storage.

4.1 RFID technology for IPS

For the needs of this project, we use RFID passive tags. Rather than tracking accurately all the participants, we focus only on the social activity that happens around stand-up tables. This ensures random movement and co-presence of people is not recorded by the system. Furthermore, the participants are aware that they can be tracked within the range of the tables, and not outside of these. Therefore going "off the grid" is possible, reducing the feeling of being monitored. With passive tags, no maintenance is needed and the read range going up to 16 meters allows for various types of applications (beyond social interaction, but still indoors). In the next chapter, we present the general architecture of the system we designed to track participants during an indoor social event.

4.2 RFID hardware

The range of RFID hardware is wide. Since the technology is market ready and well spread, many manufacturers propose their own solutions. Each solution fits different applications due to their properties such as read range, read rate, tags types, provided API, and prices. In our case the read range and rate are the priority, then the price for developing the prototype. As previously explained, the RFID technology is mainly applied to merchandise tracking and very specific applications related to it such as storing and retrieving packets. In this section we discuss the needs the RFID technology has to fulfil to become an IPS and the selected hardware.

4.2.1 Hardware selection criteria

In order to select hardware that corresponds to the needs of ITSI, it is essential to establish a list of requirements and use cases the technology should support:

Passive tags: For our application the participants should wear as little technology as possible. Therefore asking participants to wear an RFID reader is not a viable option. Hence the participants have to wear RFID tags which are smaller and more discrete. As explained before, RFID tags come in two flavours: (1) active or (2) passive. Active tags require batteries and are more expensive, yet their read range is larger. To avoid mid-event energy shortage on a tag, we choose to use passive tags.

Read range: Since ITSI uses passive tags, the RFID readers need to have more powerful emitters and higher gain antennas to make up for the read range. In our case, two types of read ranges are required to either record participants at the networking tables and at the buffet tables. A reader at a networking table does not need more than 100 centimetres read radius whereas the buffet tables need more.

Read rate: For data sampling reasons and for convenience, the readers need to record the tags within their read range at a 1Hz frequency. To make sure all the nearby tags are recorded, the read rate should be high enough to identify every second up to 15 or more unique tags.

Connectivity: RFID readers mainly use interfaces such as serial, serial over USB, RJ45 or wireless medium to communicate with a controller. For the prototype, we decided not to rely on wireless communication as power supplies would be needed for each reader. The serial choice is also discarded as adapters are required to connect to most devices.

API: Since the RFID readers are used in a custom software, they should come with an Application Programming Interface (API). This API has to be compatible with the programming language used to create the controllers and work. Verifying whether an API is actually working requires both hardware and time.

Reader size: Although not as important as the other criteria, the readers' size matters as they should be hidden as much as possible from the users.

Among these criteria, the API and read range are the most important. Finding RFID readers that can read passive tag from a good distance and that are not overly expensive requires thorough research. Moreover in absence of a working and well documented API, creating the controlling software becomes tricky and time consuming.

4.2.2 Hardware choices

To build the first part of ITSI, hardware has been selected using the aforementioned criteria. We chose the THINGMAGIC USB PLUS+ RFID Reader as shown in figure 4. This reader is capable of reading and writing UHF Class 1 Gen 2 (ISO 18000-6C) tags, which are very common and have good performance. Depending on the tags used, the read range may reach up to 1 meter and read up to 200 tags per second, which is enough for the networking tables. From a connectivity perspective, it uses an emulated COM port interfaced with USB 2.0. This allows to connect virtually as many readers as desired to a single host. Power supply and data interface are both handled via the USB port. Finally, the reader size is 97 x 61 x 25 mm, which is small and convenient enough to be put in key locations to track participants.

Figure 3: THINGMAGIC USB PLUS+ RFID reader

This RFID reader comes with a SDK containing a demonstration software, the Mercury API[8], and

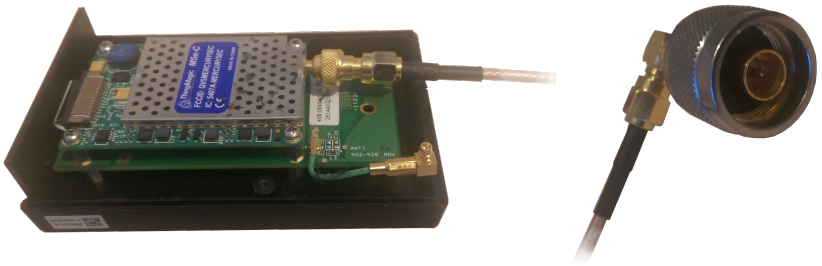
[8] http://www.thingmagic.com/index.php/mercuryapi, (October 2014)

sample codes on various development platforms and programming languages. The Mercury API supports the following application types:

- .NET applications in the .NET Compact Framework v2.0
- Windows applications in the .NET Framework
- Windows applications in the Java Framework
- Linux (Intel) and MacOSX applications in the Java Framework
- Android applications in the Java Framework
- iOS application with Mercury API using Xcode framework

This RFID reader has a price tag of 445$ per unit. The variety of supported application types makes it a relevant choice to ensure reusability of this hardware in other developments. The reader is equipped with an internal linear polarized antenna with peak gain 1 dBi in the range 860960 MHz. For our use on networking tables the read range is sufficient, but not for buffet tables. In order to increase the read range of the readers that equip the buffet tables, the readers had to be modified. For these very modifications, we mandated the help from the CSEM[9]. They have disabled the internal antenna, and the cable linking the controller to the antenna has been made longer and exiting the reader's plastic case:

Figure 4: Reader modification: custom external antenna connector



Using this new external connector, new antennas have been tested and the read range increased to almost 15m in direct line of sight. Considering the greater size of these antennas, they should be placed under the tables to hide the technology from the sight of the participants.

To be used in association with these RFID readers, we used ZEBRA UHF RFID cards that operate in the frequency range 860-960 MHz. The form factor is the same as any credit card and they can be read up to 15 meters from the readers.[10]

4.3 Readers controller

The readers controller is the piece of software that handles connection, disconnection, and read commands on the RFID readers. The original concept for the controller involved plugging an RFID reader via USB to an Android tablet. The tablet is used as a power supply for the reader and relays the records captured by the reader via its built-in Wi-Fi to the server. Since the RFID readers API provides code and application samples, we installed the official Android application, connected the reader to a tablet and went through a first batch of tests. The Android tablet never succeeded in connecting to a reader to take control of it. ThingMagic replied to our request for support that their official Android code has issues with the FTDI USB virtual COM port driver. This means that the adapter from the native serial interface of the reader to external the USB's is not supported. However the new readers in production have native USB support and are correctly supported by the API, therefore, only with M6e processors and higher. As the RFID readers we have feature the M5e, they are not supported. To solve this issue we chose to fall back to the Java framework on a PC. The first tests were immediately conclusive and operations such as connection

[9] http://www.csem.ch/site/default.asp

[10] Specification sheet: http://rfid.atlasrfidstore.com/hs-fs/hub/300870/file-1524324810-pdf/Tech_Spec_Sheets/Zebra/ATLAS_Zebra_UHF_RFID_Cards.pdf, (March 2015)

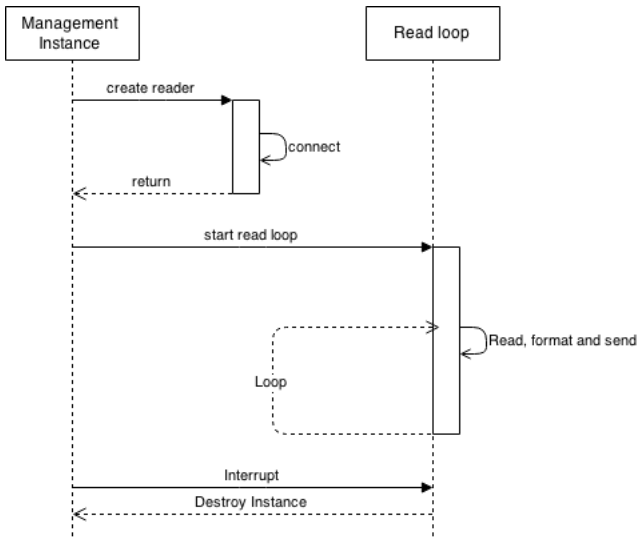
and reading tags worked flawlessly. We developed a first prototype capable of connecting to a reader and perform tag reads every second. We then implemented a multi-threaded version that creates one thread per reader the application controls.

4.3.1 Setup and data collection

Since attempts to connect the readers to Android-based hardware have proven unsuccessful, a replacement solution has been designed. This solution requires the connection via USB cables of all the readers to one or many hosts for them to be managed. The USB cables handle both power supply and data transport. Since the readers use a FTDI driver to emulate a virtual COM port, then as many readers can be connected to a given host as the operating system supports. In our case 14 readers have been connected to the same host. We used daisy chained USB hubs to connect all the readers together and a combination of USB cables of 7.5m and 2m.

The readers controller uses a configuration file that stores a mapping between the readers virtual identifiers[11] in the database and the virtual COM port assigned to it by the operating system. The mapping in the configuration file is created by connecting one by one all the RFID reader to the USB network and by assigning its respective virtual reader identifier to it. The readers controller will automatically load the configuration file when launched and stands ready to create a management instance for each reader found. Each management instance is created as a thread running in a thread pool. These threads follow the flow shown in figure 5. First the respective arguments from the loaded configuration file are forwarded to the new reader-attributed thread that stores these data and attempts to create a handle to the RFID reader by connecting to an address formatted as "tmr:///" followed by the virtual COM port.

Figure 5: Flow chart of management instance



If the connection process is successful, a handle is returned and stored. Using this newly acquired handle, the thread can start communicating with the physical reader and sends a the reader a request to perform a read operation (400ms) and report back an array of tags read. Each cell of the array contains the serial number of the tag read and its RSSI. The data thus acquired are then formatted and sent to the web server, in accordance to the description in chapter 4.3.2. The thread waits for 600ms more and repeats the reading process at 1 Hz.

4.3.2 Reading, formatting and sending

In figure 5, we introduced the iterative reading process of ITSI. Each second the management instance requests the reader it handles to perform a read operation from which an array of tags with their respective

[11] "Identifier" field in the sensor table of the database.

RSSI is returned. The reader controller then stores all this information in a "ForwardData" Java object. The ForwardData object stores the following information:

- **Time Stamp**: time of recording.
- **RFID reader identifier**: references to which reader the values have to be attached in the database.
- **Event identifier**: references the event that these records contribute to.
- **Array of records**: associativity table containing the RFID tag serial number and its RSSI, for every tags detected and read during the current time stamp event.

Once the data are stored in the aforementioned data structure, it is read and formatted to a JavaScript Object Notation[12] (JSON) file. This file is then attached to the body of an HTTP Post request. JSON is a lightweight text format designed to be easily read by both humans and machines. Finally, the HTTP Post request is sent from the reader controller to the web server to store the newly acquired data from the readers. The way these data is received and processed by the web server is described in chapter 5.5.

[12] http://json.org/, (February 2015)

5 Server-side middleware

In this project, the system has to be able to communicate with a broad range of technologies. Whether it is accepting data from the readers or distributing processed data to any type of client — web-based, native or mobile applications. To make communication between the RFID readers and the clients possible, we use an all-purpose interface: a web service. The web service is a middleware that manages the communications between all the elements of ITSI, namely the RFID readers, the database and the clients.

5.1 Database choices

Choosing a technology for the web service requires to know which type of database suits the needs best in terms of reactivity and data throughput. There exist two types of databases: (1) relational and (2) non-relational. In the first case, a relational database is the most common type of database because it is intuitive to create and query. They operate on a mathematical basis with the set theory. The main advantages of relational databases are their intuitive way of making queries and the space efficiency as redundancies are limited as much as possible. The downside of such databases is the number of simultaneous queries that may be carried out. Reading operations no not pose any issue but writing operations can become problematic if a high number of concurrent storing queries are made. In the second case, non-relational databases may use multiple types of underpinning mathematical theories. The most obvious difference with relational databases is data redundancy. While relational databases needed to be storage space efficient, non-relational databases don't have to. Redundancy is actually what makes the non-relational database interesting as it can cope well under highly simultaneous operations without causing data and/or concurrency hazards.

For this project we focus on low scale events with an expected participation of up to 100 participants. The readers transmit data through the reader controller at a 1Hz frequency, with a theoretical maximum of 120 participants detected. This makes at most 100 records per second in the database. On top of that, clients can make queries on the database for live visualizations, data crunching or to compute statistics. For this use, concurrent write operations are relatively limited and of low spatial complexity. For this prototype only one client performs requests on the web service at a given time. This also considerably limits the amount of queries the database has to handle. Therefore using a relational database becomes a valid choice as performance is good enough to deal with the maximum theoretical throughput expected from both the readers and the clients. Also, the reduced complexity of designing a relational database makes it easier to understand for third parties potentially involved on the current project. Note that using a non-relational database would also be possible but we defined that this project is a proof of concept and that a relational database already fulfils the requirements. In the event where the size of the events and or the number of client applications requesting data from the web service were to increase, then a

non-relational database would be required.

There exist many types of relational and non-relational databases. We will focus our attention on describing the most popular relational databases and explain their advantages and disadvantages. Among the most popular relational databases, the three major ones are the following[13]:

- SQLite: text-based, application-embedded and powerful.
- MySQL: the most popular and commonly used.
- PostgreSQL: the most advanced, SQL-compliant and open-source.

The first option is **SQLite**. This database is embedded into the application itself, which makes it very fast and efficient — but accessible by only one client. The database is contained into a single file, which makes it easier to backup and share than any other database system. SQLite is mostly based on the SQL standard except for a few operations, which also standardizes the application as it has to respect the SQL rules in its queries. The biggest advantage of SQLite is its ease of use for development and testing. Most developments need a solution that scales well with concurrency but remains easy to use for prototyping purposes. On the downside, SQLite does not offer user management and remains limited performance-wise as only a single writing operation can be carried out at a time.

The second option is **MySQL**, which is the one of the popular and large-scale relational database system. Unlike SQLite, MySQL uses a server and a daemon process to access the database. The advantages of MySQL are its support and documentation, the number of features included, the security features and its performances. Among its disadvantages we find its slow development speed due to a more complex setup process and steeper learning curve, its concurrency limitations for read-write operations and the partial compliance to the SQL standard.

Finally, the third option is **PostgreSQL**. This database is standards-compliant and extensible. Compared with the two previous database systems, PostgreSQL is designed for performance and allows for concurrent read-writes operations to be performed thanks to the implementation of Multiversion Concurrency Control (MVCC). It also supports extensions such as complex stored procedures, which can alleviate the work done in the applications using the database. PostgreSQL is therefore very extensible as it can be extended programmatically. Additionnally, this database system has strong community support and readily available third-party tools to help working with it. As for the disadvantages, PostgreSQL has performance issues for read-heavy operations. Another issue is the hosting of these databases is quite hard to find as by nature this database system is harder to use and put in place than the aforementioned ones.

For ITSI's design, we chose to use SQLite. The performance requirements of the prototype are quite limited and the most important is the ease of deployment and testing. SQLite requires no installation or server since it is self-contained and embedded with the application, which makes it the fastest to get started with according to table 7. The fact that no user management is natively implemented is not an issue since the web service may rely on a separate database for this purpose.

Table 7: Relational databases comparison

	User management	Concurrent write operations	SQL standard	Use complexity
SQLite	No	Supported	Partial	Fast and easy
MySQL	Yes	Partial	Partial	Easy
PostgreSQL	Yes	Supported	Complete	Complex

Therefore we conclude that SQLite — despite being lightweight — is a complete and sufficient database system for our prototype. Note that it is not a definitive choice but a relevant one for development purposes. In the case where more concurrent read or read-write operations are required, then there would be no choice

[13] https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems, (October 2014)

but to use a non-relational database. We will further discuss this option in chapters 5.2 and 8.3. There also exist multiple types of non-relational databases [14].

5.2 Web service technologies

The database is the first step in choosing a technology to create the web service. Web development frameworks come in a multitude of programming languages with different learning curves, installation process, set of libraries, etc. For this project, flexibility and ease of modification, and upgrading the work already done is of the essence. Therefore the first feature the framework should implement is Object Relational Mapping (ORM). ORM allows for the creation of an object-like hierarchy of data that can later be applied on a database. This makes the development efficient and quick for third parties to get used to. Moreover it allows to change the database system in use without altering the model. Web frameworks that have ORM capabilities are CakePHP, Django and Ruby. The second feature the web framework should implement is the availability of plugins. Creating features and functionalities from scratch is both time-consuming and redundant when relevant plugins already exist. CakePHP and Ruby already have the bundled JavaScript library included, but for this project only require a few modules that can easily be added manually to any of the three aforementioned web frameworks. Finally, one of the most important feature is the community support. With it comes plenty of documentation, tutorials and help to create new applications.

In this application, we mainly focus on managing lists of data, sort through and arrange them for proper use. For list comprehension, Python is a known programming language because it packs a specialized and efficient syntax. Moreover Python is largely used for prototyping applications, allowing us to create more in less time. Thanks to its simple and short syntax, Python also makes it easier to pick up an already existing project. We therefore choose to use Python as a programming language. However the range of Python-based web frameworks is wide. Whether it is Django, Flask, Pyramid, Tornado, Bottle, Diesel, Pecan or Falcon, they all serve various types of applications. We discuss the three most common ones: Django, Flask and Pyramid.

Flask is the most recent of the three frameworks. It focuses on small scale projects as it is a "microframework". The advantages of Flask are its setup simplicity and quick deployment. Being lightweight, most of its components have to be added externally, which gives a lot of opportunities for customization and extensibility. However the community around Flask is small and support on the web too. Flask is the Python framework to work on small yet fully customizable applications. In opposition to Flask, Pyramid and Django are designed to be able to create larger applications. Flask does not provide any boostrapping, which in the context of web frameworks is a starter application and website with pre-configured content. Starting from there a developer can customize the bootstrap into the application he/she wishes to create. In Pyramid's case, external modules can be implemented within the core application, which enables flexibility in the projects. Django enables flexibility by providing modules the developer can chose to use or not. Django is also the most popular of the three web frameworks. In table 8, the most important selection criteria have been summarized[15]:

Table 8: Django web frameworks comparison

	Flask	Pyramid	Django
ORM	Via libraries	Via libraries	Native support
Target	Small applications	Large applications	Large applications
Boostrapping	Not included	Built-in	Built-in
Templating	User-Defined, Jinja2	User-Defined, Chameleon	Not User-Defined, Built-in
Community size	Medium	Medium	Very large
Year	2010	2005-2006	2005-2006

In our conclusion, Pyramid is the most flexible framework, and it is worthy to note it powers websites such as Dropbox. It is capable of handling projects of all scales and leaves most of the technological

[14] https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models, (October 2014)

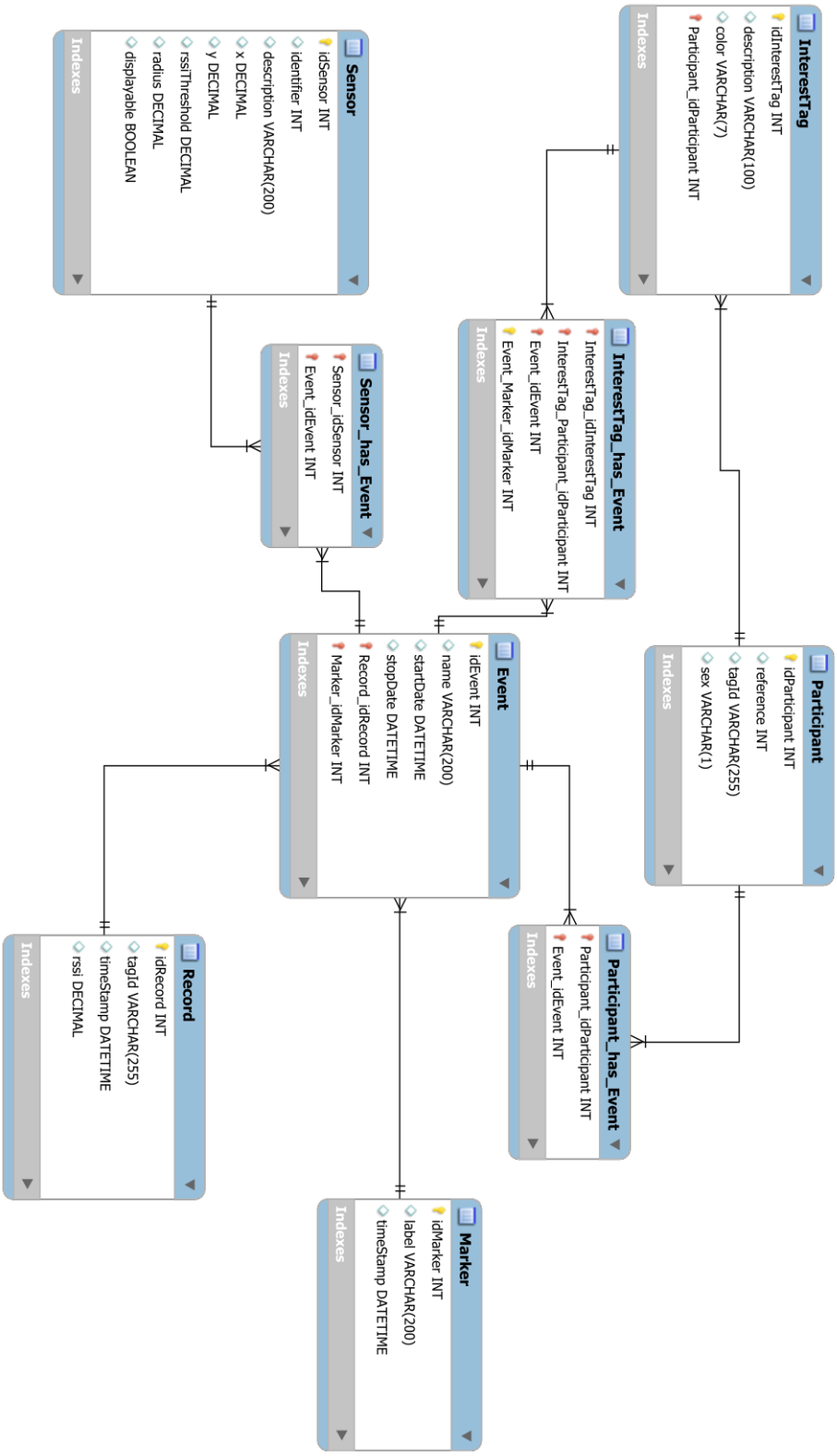
[15] https://www.airpair.com/python/posts/django-flask-pyramid#1-introduction, (October 2014)

choices to the developers. Flask definitely illustrates itself on smaller projects because it is quick to set up and is usually used to power small tools. Finally, Django is the most popular of the three frameworks and powers websites such as Bitbucket, Pinterest or Instagram which handle large throughput and high availability. It readily packs all the required components by default, including an ORM. The built-in, off the shelf ORM and vast support from the community turns the tide in favor of Django rather than Pyramid.

By taking into consideration both database and web framework choices, we decided to use Django with an SQLite database to develop the first prototype of ITSI. In the following chapters, the database schema and implementation are presented as well as the web application. We detail the methods of the web service and explain their best use.

5.3 Database diagram

Figure 6: Database schematic from Django model



5.4 Database models

As previously explained, Django is packed with an object/relational database creation and management tool. Django management offers the possibility to automatically apply the object model to a database. In this section we describe each object of the database model and explain both their composition and the purpose they fulfil. The complete database model is given in Appendix II [9.1].

5.4.1 Interest tag

Interest tags represent the interest of a participant. Each participant may have one interest tag that adds extra information such as a textual description of the interest as well as a colour for the visualization. Thanks to the interest tag, multiple participants can share the same interest and can be represented using the same colour.

Table 9: Interest tag database model

Field	Type	Description
description	models.CharField(max_length=100)	The description of the interest tag
colour	models.CharField(max_length=7)	Hexadecimal colour value

Note that the colour field must be of the form "#" followed by the hexadecimal colour tag. We chose to leave the # at the beginning of the field for compatibility reasons as many web-based applications directly use such a colour encoding.

5.4.2 Participant

The participant object represents a user of the indoor tracking system. A participant is made of a reference which acts as an identifier, a unique RFID tag identifier, a gender for research purposes and an interest tag. The reference field is used for readability and ease of query in the database. The gender field accepts a single char which assumes the value "M" for male, "F" for female or "U" for unknown in the case where this information is not available.

Table 10: Participant database model

Field	Type	Description
reference	models.IntegerField(unique=True)	An identifier for a participant
tagId	models.CharField(max_length=255)	Identifier of the RFID tag
gender	models.CharField(max_length=1)	Gender of the participant, one character
interestTag	models.ManyToOneField(InterestTag)	One-to-many key to Interest Tag

For anonymity reasons, we chose not to include any other data such as names, phone numbers and addresses from the participants. Therefore the system remains anonymous to avoid breaching the privacy of the participants. Using the reference field, it is always possible to tie the list of participants to an external database which contains these personal data.

5.4.3 Sensor

A sensor represents an RFID reader. The sensor object fulfils two main roles in the system: (1) it physically describes the reader and (2) it allows to store preferences that are used in the visualizations. First, each reader may be different as they are placed in different locations or might exhibit different detection radii. We therefore store for each sensor the X-Y coordinates for localization and a radius representing the read range. With this implementation, it is possible to either have two readers at the same location but with different read ranges, or to have a single reader performing multiple reads with various read ranges and that stores its data under two logically different sensor objects.

Table 11: Sensor database model

Field	Type	Description
identifier	models.IntegerField(unique=True)	Identifier of the sensor
description	models.CharField(max_length=200)	Description of the sensor
x	models.FloatField()	X coordinate of the sensor
y	models.FloatField()	Y coordinate of the sensor
rssiThreshold	models.FloatField()	Individual threshold in dB for read range
radius	models.FloatField()	Read range of the sensor
displayable	models.BooleanField(default=True)	Whether this sensor should be displayed or not

Second, the readings of a reader will be reused to populate the live visualization. For that purpose we provide a couple extra fields: the rssiThreshold and the displayable fields. The RSSI threshold allows us to set a minimum RSSI the readings should feature to be displayed on the live visualization. Finally, the displayable field allows us to distinguish between a sensor that should participate to the live visualization and one that does not. It can be interesting for analysis purposes to have RFID readers perform sampling at different rates or record in larger read ranges to perform multilateration.

5.4.4 Event

The event object is an administrative entity. It hosts basic information about the event itself and gathers all the interest tags, sensors and participants tied to a given event. The basic information features the name of the event, a start and a stop date. The rest of the fields are many-to-many relationships with the interest tags, the participants and the sensors. Hosting this extra information makes it easier to query for all the potential interest tags during an event, all the involved participants in the event and all the sensors sending information about the whereabouts of the participants. Furthermore, such a structure allows for the creation of security mechanisms against RFID tags or sensors sending information to the web service without being registered in the event — detection of fake data injections. These erroneous input feeds can then be discarded to avoid data poisoning.

Table 12: Event database model

Field	Type	Description
name	models.CharField(max_length=200)	The name of the event
startDate	models.DateTimeField()	Start date of the event
stopDate	models.DateTimeField()	Stop date of the event
interestTags	models.ManyToManyField(InterestTag)	Possible interest tags of the event
participants	models.ManyToManyField(Participant)	Participants of the event
sensors	models.ManyToManyField(Sensor)	Sensors of the event

The start date proves useful when used as a starting point for statistical computations. It allows to align all computed statistics in time, hence making them visually clearer.

5.4.5 Temporal marker

Temporal markers serve two purposes: (1) they allow to record extra information. Such information can be stages during an event. (2) They serve as markers for graphical representations of the data. A marker can be assigned to only one event.

Table 13: Temporal marker database model

Field	Type	Description
label	models.CharField(max_length=200)	Description of the marker
timeStamp	models.DateTimeField('timeStamp')	Date of the marker
event	models.ForeignKey(Event)	One-to-one key to an Event

5.4.6 Record

A record is a piece of data gathered from an RFID reader. A record contains a single tag reference with a time stamp. The time stamp is the time at which the tag was detected within the read range of a given sensor. Since that tag was read, the reader returns an RSSI value as well.

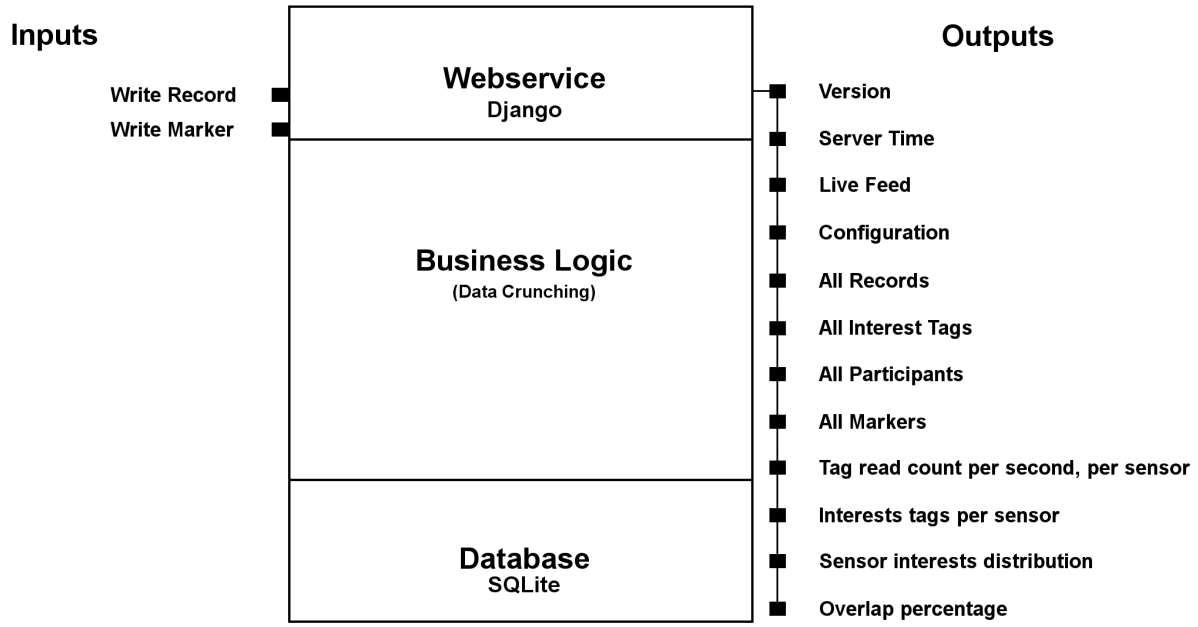
Table 14: Record database model

Field	Type	Description
tagId	models.CharField(max_length=255)	Tag identifier of the record
timeStamp	models.DateTimeField('time stamp')	Date of recording
rssi	models.FloatField()	RSSI value of the record
event	models.ForeignKey(Event)	One-to-one key to an event
sensor	models.ForeignKey(Sensor)	One-to-one key to a sensor

5.5 Web service interfaces

To bind the database to potential client applications, we implemented a Django web service with a set of input and output methods. Rather than allowing clients direct access to the data base and other features in a non standardized fashion, we formatted a set of methods all clients may call. Figure 7 represents all the available methods offered by the web service.

Figure 7: Input and output methods of the web service



In this chapter we describe each method and its inputs and outputs. Each description contains the request type, the input parameters, the returned values, the use cases of each method and its URL scheme, respectively. Each method natively supports URL callbacks to avoid HTTP cross-domain limitations.

5.5.1 Input methods

Write Record POST method that stores a record in the database. A record is accepted and saved in the database if the HTTP POST request packs a JSON file. The JSON file stores a time stamp, a sensor identifier, an event identifier and an array of (RFID tag serial, RSSI) pairs. To reduce message size, we chose to pack all the records coming from a single reader spanning over a second together. Therefore the JSON file input in this method may contain one to multiple records. Each record is then

stored individually in the database. Through this process we aim at reducing communication overhead. The method returns an "OK" message if the operation was successful, and a "FAILED" message otherwise.

URL: http://<ip-address>/hubnet/irec

Write Marker POST method that stores a marker in the database. The methods takes one input parameter "label" which is the textual description of the marker. The markers are used to create a time marker during an event. It becomes convenient to split data between two different times during an event. In our case, we use it to isolate the data for statistical computations only focused on a given part of the event, and for graphical representation of these time frames.

URL: http://<ip-address>/hubnet/mrkr/<label>

5.5.2 Output methods

Version GET method that returns the version of the web service. The version is returned as a string. Since this method is lightweight, it can be used to perform RTT measurements.

URL: http://<ip-address>/hubnet/version

Server Time GET method that returns the clock time of the server. This function is only used for synchronization purposes between the web service, the RFID readers and the client interfaces. This call is usually made before requesting a set of records to get live updates. Note that this function has one parameter: "diff". This parameter is the time difference in seconds we request from the server. A difference value of 2 means that the server time will be returned minus two seconds.

URL: http://<ip-address>/hubnet/time/<diff>

Live feed GET method that returns a list of records that were received and stored in the database for the event which identifier is equal to the parameter "eventID", and for which the time of record matches the parameter "timeStamp". This method is used to update live visualization on client interfaces. It is advised to regularly perform queries on the server for its time value to make sure the data remains synchronized. Any client interface that requires all records at a given time for a given event may call this method. To make clients like sliders to visualize the data through time, it is recommended to use the "All Records" method.

URL: http://<ip-address>/hubnet/lvf/<eventID>/<timeStamp>

Configuration GET method that returns all the sensors stored in the database for a given event specified by the parameter "eventID". This method should be used to gain access to data related to the sensors or, in our case, automatic drawing of the tables in the live visualization. Indeed, each sensor object contains information about its X-Y coordinates and the displayable field.

URL: http://<ip-address>/hubnet/cfg/<eventID>

All Records GET method that returns all the records stored in the database for a given event specified by the parameter "eventID". This method should be used for external data crunching and processing. For live visualization it is not recommended to use this method as the overhead of data can be important and tamper with the responsiveness of the visualization through latencies.

URL: http://<ip-address>/hubnet/arec/<eventID>

All Interest Tags GET method that returns all the interest tags stored in the database for a given event specified by the parameter "eventID". This method should be used to get all the possible tags, their description as well as the hexadecimal colour value associated with them. The purpose of this method is to allow to configure rendering settings for visualizations. In our example, the live visualization creates

the legend of colours using this method.

URL: http://<ip-address>/hubnet/atag/<eventID>

All Participants GET method that returns all the participants stored in the database for a given event specified by the parameter "eventID". This method should be used to gather a list of the participants to an event. For any use ranging from search index to live tracking.

URL: http://<ip-address>/hubnet/apar/<eventID>

5.6 Data crunching

All Markers GET method that returns all the time markers for a given event. Each marker is made of a label and a time stamp. The time stamp is the difference in seconds between the time of the marker and the start time of the selected event.

URL: http://<ip-address>/hubnet/statmrkr/<eventID>

Tag read count per second, per sensor GET method that returns an item for every second during an event. Each item is a pair containing the second number and the number of tags read on a given sensor during that second. This method is used for offline data crunching as it is expensive for the web server. To efficiently count tags per second, we create an array of size N where N is the total number of seconds in the event. We then iterate on the set of records and compute its index in seconds. The value in the array is then incremented by one at the correct index. Finally, the results are formatted for .json output.

URL: http://<ip-address>/hubnet/statdata/<eventID>/<sensorID>

Interests tags per sensor GET method that returns an item for every second during an event. Each item contains a set of pairs that associate an interest tag with an amount of tags read for each second. This method can be used to track, for each second, the number of tags of each interest. The data crunching part of this algorithm is exactly the same as for the Interest tag per sensor, except that the data for each second is split among the interests. Therefore we work with a matrix of size M x N where M is the number of interests and N the total number of seconds in the event.

URL: http://<ip-address>/hubnet/statdataTag/<eventID>/<sensorID>

Sensor interests distribution GET method that returns the cumulative number of tags read over the full event on one sensor, per interest tag. This method delivers data that specify each interest tag label and their respective count of records.

URL: http://<ip-address>/hubnet/statdataInterests/<eventID>/<sensorID>

Overlap percentage GET method that returns the number of overlapping records between two sensors, over the total number of tags read for sensor with identifier equal to "sensorID1". The input parameters are the event reference "eventID" and the identifiers of the two sensors to compare. Note that this method should be called twice: once comparing sensor i with sensor i+1 and then the opposite. The overlap count will be the same each time, but to compute the percentage we need to get the total tag reads count of each sensor and add them together. Therefore, by executing this method both ways, the total number of tags of both sensors combined can be computed.

URL: http://<ip-address>/hubnet/statdataOverlap/<eventID>/<sensorID1>/<sensorID2>

5.7 Conclusions

In this chapter, we have selected two technologies, Django and SQLite, according to what we believe to be the most important criteria to make this prototype reliable and extensible. The code listings for most of the web service methods are available in the appendixes.

6 Client-side interfaces

In chapters 4 and 5, we presented the technologies required to record participants whereabouts in real time and store/retrieve this data. Although sufficient to be considered as an IPS, it can be extended by adding graphical interfaces to make the information stand out and make it actually usable by any participant. We have designed a set of methods to gather data in the web service so that many different client interfaces may use these — to make human-computer interactions easier. As said in the introduction, this project is two-fold: (1) creating a live visualization that operates during the event itself and (2) a reporting solution with in-depth analysis of the data collected throughout the event. In this chapter we detail the live visualization, from concept to implementation. The reporting solutions are described in chapter 7.

6.1 Initial concept

Sharing and meeting with the exact people we are looking for during a social event can quickly become tricky. The reasons range from lack of visibility, too slow turn rate or not knowing what interests other participants hold. In any of these situations, time is lost trying to find the right person to talk to. This project aims at providing the tools to optimize the time of each participant by providing a clear, informative and visualization of the event as real time as possible. To increase the chances of having a fruitful interaction, we identified key information required by participants: (1) A geographic representation of interests, (2) the interests of each participant and (3) an easy way to locate oneself in space.

Figure 8 depicts the initial concept we came up with to visualize a live social event. The visualization[16] highlights the key informations aforementioned:

Room space The shape of the room as well as the screen (thicker stroke) where the live visualization is displayed are rendered to help the participants identify locations. This is only used for representation's sake.

Detection zones The four bigger disks incarnate the zones where participants are tracked. Each of these zones uses a pie chart view based on the interests of the participants nearby.

Participants Each smaller disk incarnates a participant detected in a zone. Since there isn't an accurate XY tracking of the participants, their positioning within the detection zones is arbitrary. We chose to enhance clarity by drawing them according to a linear distribution around the detection zones. Note also that the diameter of the disks may vary depending on the score of each participant. The score is used as an incentive to enhance the experience of the event and is thus highly event-dependent.

[16] *Note:* This visualization is not representative of the final product.

VIPs The blue disks could be VIP participants (talker, moderator, administrator, etc). Having a distinctive colour and maybe shape to help identifying them quickly.

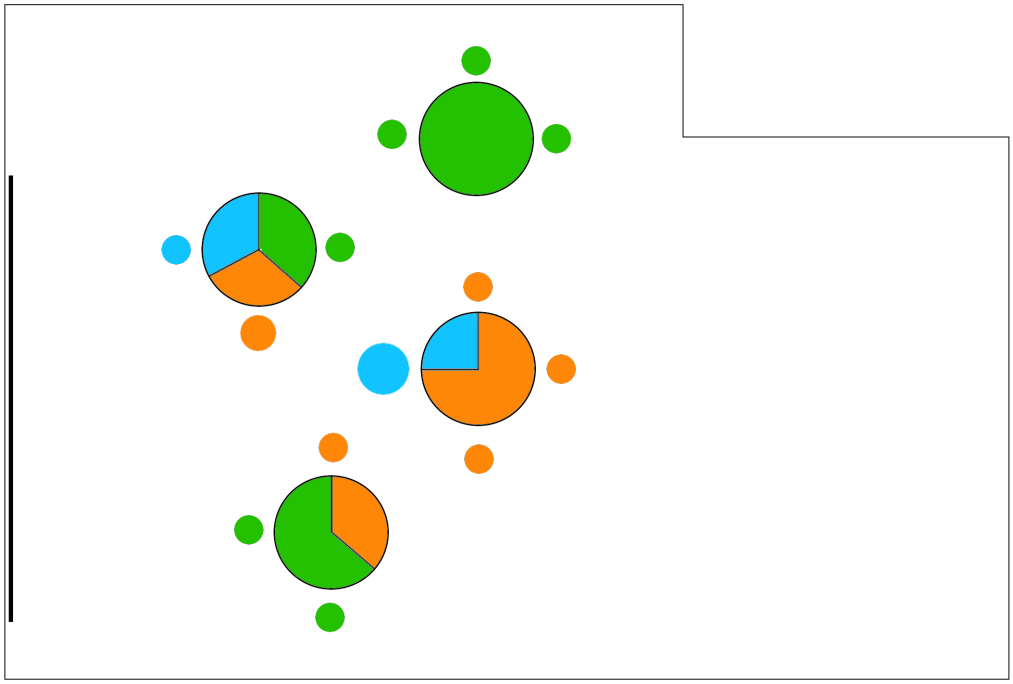


Figure 8: Initial concept for live visualization.

Our hypothesis claims that a participant would need less time to identify another participant or group of participants holding the interests he/she is looking for. The map featuring geographic locations of interest throughout the event is updated in real time and completed with reports computed from the data collected. These reports would be made accessible once the event is over.

6.2 Use cases and possible scenarii

The original need behind this project takes its roots in optimizing a social event where the participants are looking for all types of encounters. By providing global awareness of an on-going event in real-time, efforts can be focused on talking rather than finding the right person to talk to. In this section we present user-centred scenarii to demonstrate how such a system could be used. The first use case would be during a forum where participants can gather and chat:

1. "As a professional in the field of pharmaceutics, I want to be able to find potential partners to help me commercialize a new product. Therefore I look at the live visualization in the room and find out that a table has three participants interested in pharmaceutics."
2. "As a participant, I have met with a person of interest and engaged in a productive discussion. After a while the system tells me we have been a long time in a conversation and suggests starting to look for other participants as well."
3. "As a participant, I want to be able to ask questions to VIP participants. In the live visualization I can see them with a different colour and quickly identify where they are."

Another type of use case would be to help automatically gather information for statistical tests. Such tests aim at discovering if a product or object is more attractive to test subjects than another one. For such applications live visualization would have to be disabled and the tracking of participants becomes the only focus:

1. "As a researcher, I want to know how many testers have selected product A, and quantify how attractive the product is for potential customers."
2. "As a researcher, I want a tool that allows me to automatically track and count the interactions between the tested products and the testers."

For both the aforementioned use cases, the indoor tracking combined with systematic gathering of the data is prevalent. From this data two types of visualizations can be derived: (1) a live visualization to enable participants to gain global awareness of the social interactions and (2) data reporting, typically for the second type of use case.

6.3 Live visualization

6.3.1 Principles

Visualization is all about drawing information in an intuitive and comprehensive way. A live visualization should depict, as close as in real time as possible, the variations and constants in the information thus processed. Therefore there is a need for animations in our case. Indeed we intend to show variations of populations around zones of interest. The goal of a live visualization is to enable any participant to identify the activity in the room in the blink of an eye. A room is made of a set of tables around which participants may gather. The tables are positioned in a defined way, which act as landmarks in the room. Representing the tables in a visualization is like creating a map of the room. So the first data depicted is an actual map of the room which contributes to help participants acquire a sense of orientation in a room they most likely will be unfamiliar with. Each table also informs the participants on the zones where their whereabouts will be traced and recorded. This way a notion of privacy is represented through the visualization as well. Finally, the participants themselves are located and represented on this live map. The participants' representation has to convey multiple pieces of information at the same time: (1) the location, (2) the interest and (3) the interaction level of each detected participant.

The location of participants on a live map fits two purposes: the ease of self-location and the representation of the current distribution of participants in a room. Self-location is the first step in reaching a destination. In our case the destination is the participants or group of participants who have the interests we are looking for. It is the same process as finding one's path on a paper map. Indeed, first we need to locate ourself on the map, and then select a path that binds our current location to the wished destination. With a live map, identifying our own location is made easier by being able to compare the visualization to the real environment. By spotting similarities between the visualization and the real environment, a participant can rapidly associate dynamic elements such as other participants and locate his/her own position. Then, similar to a paper map, we need to select a path to reach our destination. The live visualization shows the entire room from a top perspective, allowing participants to plan their route to maximise the likeliness of meeting the person they are looking for.

As aforementioned, the live visualization helps determining one's location and identifying paths in the room that may lead to more fruitful encounters. To finish the analogy with a paper map, one must know his/her destination. This is the second type of information that participants represent on the live map. Since the location is represented by a shape at a given location, the colour is selected as a mean to convey the interest information of a participant. It is therefore possible for a participant to visualize the spatio-geographic representation of interests among the other participants.

6.3.2 Technologies

Now that the live visualization concept is defined, it requires a technology to draw it in an automatic and periodic way. Periodic redraws of frames is the job graphical engines perform and as such we need to look for one that suits the requirements of this project. These requirements are the following:

- **JSON compatibility:** The web service communicates almost exclusively in JSON. Being able to perform requests on the web service and decode the data returned are essential.

- **2D graphics:** The graphical engine should be able to draw quality 2D graphics with reasonable performances. Using 3D graphics remains an option as it does not yield any more useful information than a 2D visualization.
- **Portability:** The graphical engine should be able to run on the most common devices, such as PCs, tablets and possibly smartphones.
- **Free license:** Through this prototype we aim at continuing the development beyond this project. Therefore using only free tools and frameworks makes the access and extensions to this prototype easier.

To fulfil these requirements, we have to select a drawing engine that allows either to export the project to the different platforms aforementioned, or to a web-based one. Among the drawing engines that support many platforms, there are game engines. These are powerful, very optimized development tools that suit virtually any needs in terms of possibilities to render graphics. Free to use game engines that would be suitable for our live visualization are Unity 4.6 [17] or the Unreal Engine 4[18]. Unity offers an efficient 2D module and a completely programmable framework. It is also free and constitutes a valid choice of framework to create powerful visualizations. Unreal Engine 4, although more powerful, requires a monthly subscription of 19$[19]. Coding for Unreal Engine 4 is also more time-consuming than with Unity.

Working with game engines grants complete freedom over all graphical elements. Being very feature rich, these frameworks also are more complicated to get acquainted with and they usually require one recompilation per specific development target. To avoid unnecessary recompilations and tests on multiple platforms, another solution is web-based engines. Although slower, they offer cross-platform compatibility without any recompilation required. Most of these engines are wrapped into libraries embeddable into web pages. Through our research it became obvious that JavaScript-based engines are the most popular as they are fast to install and allow to use the full range of JavaScript functions. Also for a first prototype, using a fast iteration rate is essential as many changes in the creative direction can happen. Therefore we choose to develop the prototype with a web-based engine.

Paper.js, Processing.js and Raphaël are the three most used Javascript-based drawing libraries. They run with HTML5, which is becoming a standard many browsers actively support. Although Paper.js, Processing.js and Raphaël are very similar in the results they can yield, they use different approaches to get there. Paper.js and and Raphaël use an object-oriented model, which makes it possible to create an object and edit it's properties by using a reference to it. Processing.js, however, uses a no-objects model where elements of the visualization are not accessible through a reference. Doing so avoids memory overheads and thus runs faster. Working without any reference forces the developer to describe the drawing procedurally. Table 15 sums up the evaluation of all three drawing engines[20]:

Table 15: Web-based drawing engines comparison

	Paper.js	Processing.js	Raphaël
Technology	PaperScript	Processing script	JavaScript
Browser compatibility	IE 9	IE 9	IE 7
Mobile compatibility	Yes	Yes	iOS only
Object references	Yes	No	Yes

In the light of this comparison, we choose Processing.js as a web-based drawing engine to design the first prototype. Processing programming language is closer to what actual game engines do, making it easier to port the development of the live visualization to Unity for example. In the next chapter, the live visualization's work flow as well as selected parts of Processing code are reviewed.

[17] http://unity3d.com/, (November 2014)

[18] https://www.unrealengine.com/what-is-unreal-engine-4(November 2014)

[19] Unreal Engine 4 has been made free since the third of March 2015. This change happened after considering our options for graphical visualization engines.

[20] http://www.smashingmagazine.com/2012/02/22/web-drawing-throwdown-paper-processing-raphael/, (November 2014)

6.3.3 Processing and displaying data

To create the live visualization, we have designed a set of methods on the web service to provide all required information. Each of these methods provides a piece of data that serves a specific purpose to draw the frames:

- **Configuration method:** Building a canvas with minimum information is the first step in creating a visualization. It lays down the boundaries of the representation. We then need to establish a link between the canvas scale, form factor and the placement of the elements within it. Therefore scaling has to be computed as a mean to adapt automatically the size of the room to the canvas. Once the scale is known, we can place all the sensors on the map. Additional geographical information can be manually added to help users recognize distinct elements of the room.
- **All Interest Tags method:** When preparing the canvas, a legend is written at the bottom of the screen. This legend contains the colour scheme corresponding to the interest tags as well as a textual description of each interest. This should be drawn only once since this information is not supposed to change over time.
- **All Participants method:** To avoid unnecessary requests on the database, most of the data has to be preloaded on start of the visualization. Such data can be quite heavy and has to be performed with multiple calls. Therefore to avoid freezes in the visualization, the data about the participants is stored in an array when loading the visualization.
- **Server Time method:** For time synchronization between the client and the server, server time is requested for the periodic Live Feed methods call.
- **Live Feed method:** The live feed should be periodically requested by the client and the shapes representing users can be dynamically displayed based on this information.

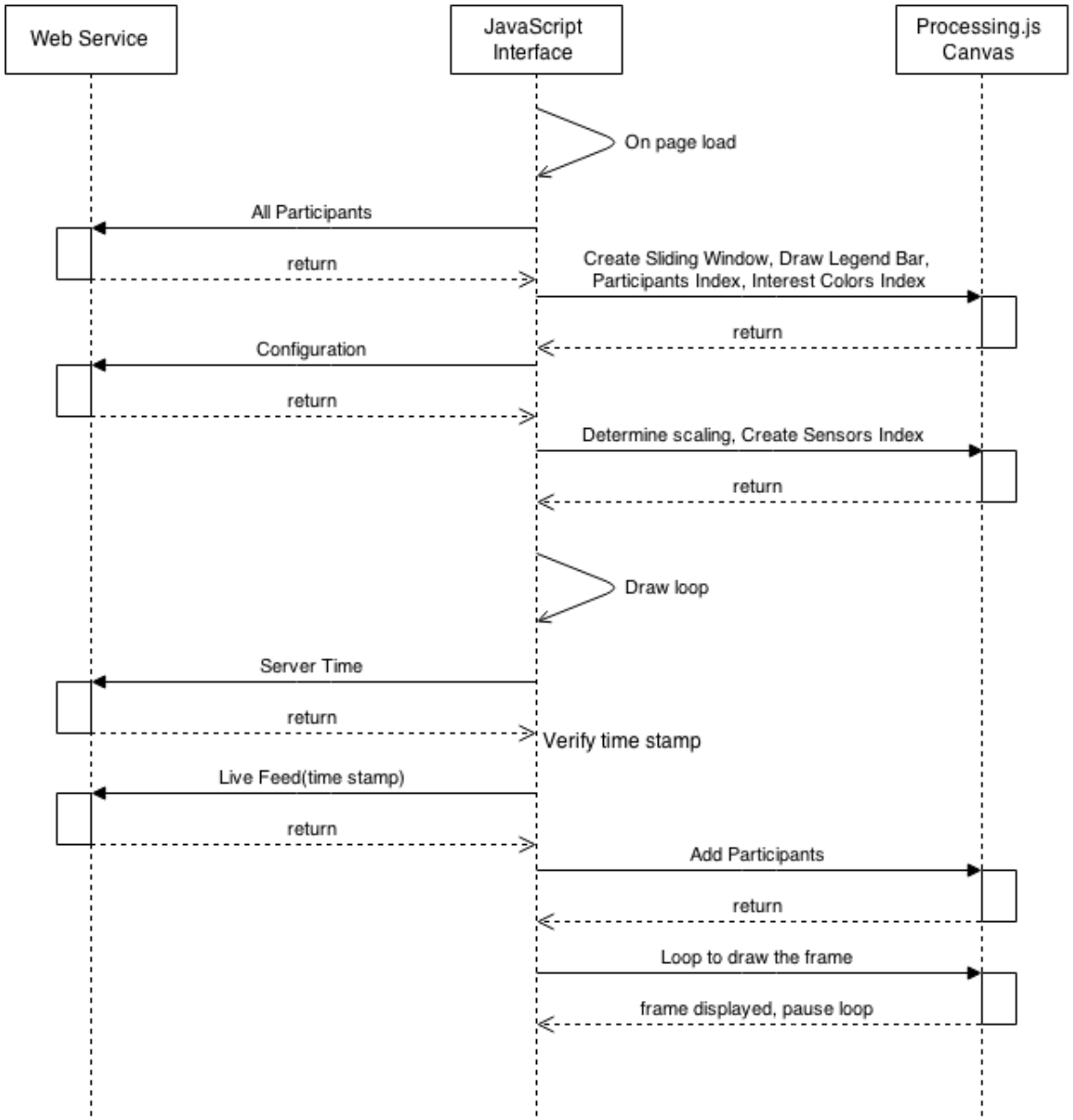
Processing.js creates a canvas embedded in a web page. The canvas allows an external JavaScript file to call functions defined within the canvas. These functions can pass data as parameters and constitute a convenient way to control the visualization from the web page directly. As presented before, the data needs to be fetched from the web service in order to create our prototype visualization. And since it is possible to call Processing.js functions from JavaScript, we use JavaScript for data management and Processing.js for displaying that information in a formatted way. Figure 9 presents the work flow and interactions between the web service, the canvas and the JavaScript file that binds these (three) modules together.

When the web page is loading, all essential data is retrieved to populate the local database. This includes the participants list and the configuration file. The participants list allows to populate an array of participants and to extract all the interest tags present in the event. Then, using the configuration file, the scale is determined by creating a rectangle that wraps the four sensors that are respectively: most left, most right, most top and most bottom. This rectangle is then multiplied by a scale coefficient that is computed by dividing the screen size by the size of the computed rectangle. With the initial configuration done, Processing.js can create the sliding window that is used to smooth the input of participants locations for the visualization. The sliding window is a matrix of M x N where M is the number of iterations (set to 5 by default) and N the number of participants. For each participant, the last five positions are kept at location M x N0, ..., N4 in the sliding window. At each frame redraw, all the values of the iterations are shifted below and N0 for each M is gathered from the Live Feed method that is called periodically. To determine the current position of a participant, the most frequent location in the five last positions is selected. In case the most recent location is 0, it means that the participant was not tracked at that time and he/she is removed from the visualization immediately.

The rendering process runs in a loop that the web page can control. Since the JavaScript code of the web page handles data retrieval asynchronously, it is best that the drawing loop does not start before all the data is received. This could lead to aberrations in the rendering of the visualization. Figure 10 shows the visualization generated from the data returned by the web service. In this case, the visualization depicts three participants around table 7.

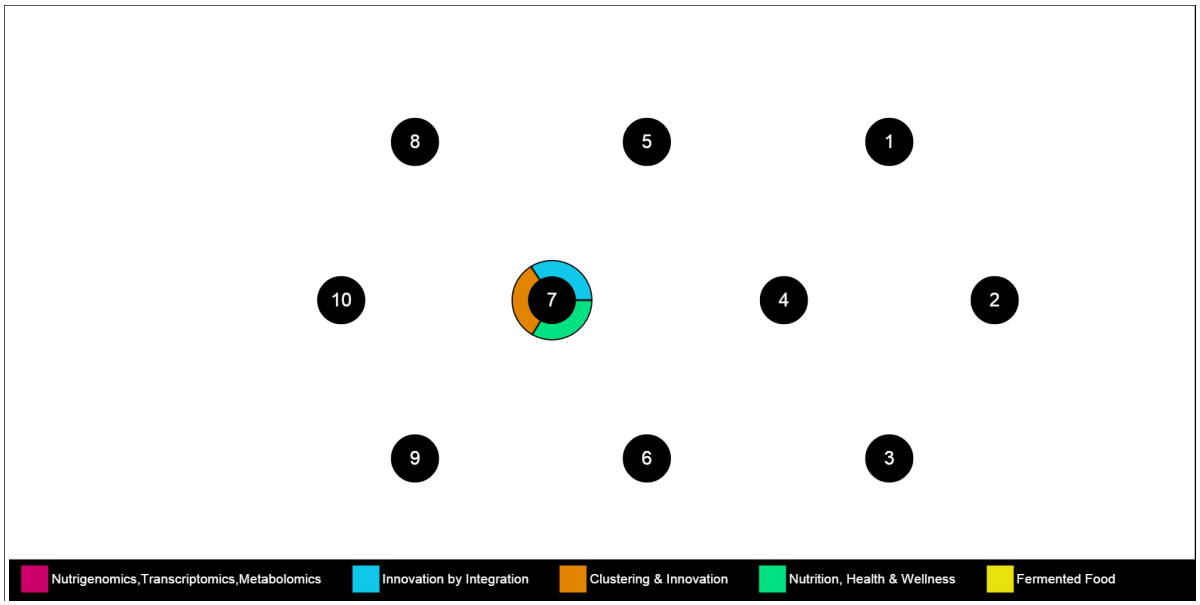
For each rendering loop, the background colour is applied, which wipes the previous content but leaves the legend bar at the bottom intact. Then, for each table, a black disk is drawn if the respective table has

Figure 9: Workflow of the live visualization



participants around it. This disk covers the radius of both the table dot and participant representations (similar to a shadow). Then all the participants are drawn: for each table, a disk is divided into X segments where X is the number of participants around that table. The order of the segments is kept by making sure that a participant drawn in the previous frame at the same table is drawn exactly at the same place in the new frame. When all participants are drawn, we cover the centre of each table with a black dot that actually represents the tables. Finally, table numbers are added in the centre of the table black dots. This concludes the rendering of a frame.

Figure 10: Generated visualization



6.4 Wizard

The wizard, as shown in figure 2, is yet another type of client interface. The live visualization example shows how it is possible to use the data collected by ITSI in real time. The wizard, on the other hand, serves other purposes such as data creation, reading, updating and deleting. This combination of operations is known as Create, Read, Update and Delete (CRUD). The wizard is used in the system to enter the parameters of the events, sensors, interest tags and participants. It should only be used to configure an event. The wizard interface is automatically built based on Django default administration panel. It is therefore generated from the database models and offers all the required database operations. A separate configuration file allows to choose the tables and fields of the database displayed in the administration panel, in which order and with which fields being considered for user research.

Figure 11: Illustration of the wizard

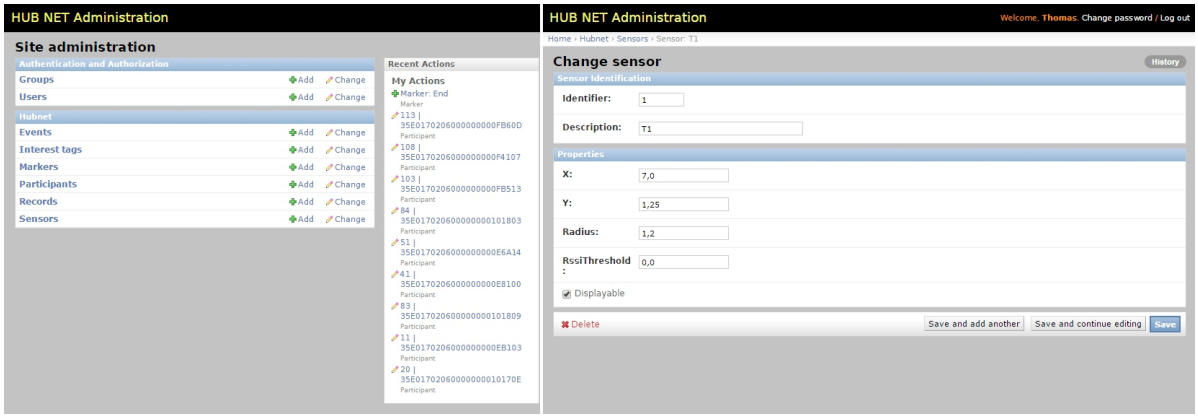


Figure 11 shows the complete administration panel (left), and the detail of a sensor creation/modification form generated from the Django database model.

6.5 Conclusions

We presented two types of client interfaces that can take advantage of the data collected by ITSI. The live visualization is the most interesting as the participants benefit from the system directly. Making a live

visualization is possible and there exist many technologies that can enable such types of visualizations. The second type of client interface is the wizard. It represents all the business operations the database supports and in this project we use it to setup the system. The third type of client interface examples we present in this project — reporting — is explained in chapter 7.

7 Performance tests and validation

In this section we test and analyse the performance of ITSI we designed and implemented. With ITSI reaching alpha state, it is important to stress test each tier of the system. Indeed, the system cannot be considered reliable if the database cannot manage the flow of data to store and retrieve, if the web service is unstable or the live visualization does not work. In the following chapters, we describe the designed test environment and the ways ITSI has been tested. We establish a set of hypotheses based on actual observations done during the test. We then present the data gathered by ITSI and show through graphical visualizations that they validate our hypotheses, thus reflecting the observations.

7.1 Experiment environment

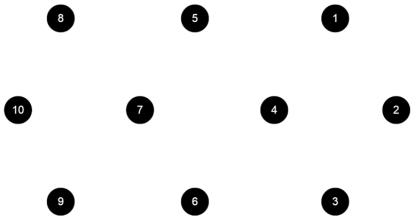
The experiment environment was provided by the SICHH. The SICHH organizes social events featuring talks and networking events among the participants. Unlike regular forums where talks are given over the complete duration of the event, the SICHH's events are limited to 10 minutes per session. However, after each session, another 10 minutes is given for the participants to engage in networking and social interactions. During the complete event, a buffet is at the disposal of the participants. The setup of the room hosting the event has to accommodate both networking sessions and the buffet. To this end, two buffet tables are set on each side of the room. The stage for the talks is set to the front of the room. Finally, the area between the buffet tables and the stage is used to host the participants and the networking sessions. This area features 10 round tables.

From a hardware point of view, the installation consists in one stock RFID reader per networking table and one reader with extended antennas per buffet table. Each networking table has a small range of tag detection to avoid overlaps between tables. However the buffet tables have a much larger detection range to completely cover the area around the foods. Each RFID reader is powered by two USB 2.0 cables connected to USB hubs. We used five self-powered USB 2.0 hubs with 7 USB ports each. Three hubs are directly connected to the computer controlling the readers. The two last hubs are connected to other hubs to increase the distance separating them from the computer. This setup allowed us to cover the whole room with as few USB cables on the ground as possible.

7.2 Validation

In this section, we focus on proving whether ITSI is able to verify by itself the hypothesis we deduced from the observations done during the event. Rather than raw data, graphics have been created to visualize data and highlight their meaning. The data used for these graphics is directly extracted from the database. The validation of the data recorded is done in two steps: (1) from a networking experiment and (2) from a research experiment perspective with the food tables.

Figure 12: Left: Networking tables layout. Right: picture of the readers on the networking tables



7.2.1 Observations

In this section we relate the event and observations made at the time. ITSI has started to record with the opening presentation made by the organizers. This first presentation also introduced ITSI and the participants started to play and experiment with it. However, the visualization experienced large delays, requiring about 20 seconds to display a participant on the visualization. Therefore the participants stopped using the visualization quite early in the event. Never the less the visualization ran during the whole evening except for the last part of free networking. ITSI was still recording though. After the introduction, the first talk has been given. The participants were still picking up food from the buffets but then paid attention to the talk. When the first networking session happened, many participants picked up plates, filled them with food and went to a networking table to discuss with other participants. We could still see some participants trying to play with the live visualization. This process of giving a talk and then allow the participants discuss among them has been repeated four times in total. However, the last talk was directly succeeded by the last presentation made by the organizers. They introduced the buffets and their difference and then the rest of the event was free networking. After that participants gradually left the event until the end. We stopped ITSI only at the very end of the event after a bit over three hours.

When observing the iterative process of having a ten minutes talk and then ten minutes of networking, trends appeared: first the participants would gather closer to the stage when a talk was being given. After the talk, the participants would usually take up on the opportunity to refill their plate and then find a place in the networking area.

Overall the RFID readers experienced more issues than expected. First the participants turned out to be more mobile than anticipated as they did not stand right in front of the tables to discuss. Having each participant around a table eases the detection of tags as they are correctly oriented to optimize the read range. Many participants stood in front of each other but next to tables, reducing the read range by orienting the tags in a configuration of decreased gain. Another difficulty tied to the same issue happened when a talk was given. Many participants would face the stage, hence blocking tag reading with their bodies. As for the other participants, they stood straighter than during a networking session. Since the RFID tags were attached to their chest's clothing, the tags did not face the readers while the participants stood straight.

7.2.2 Hypotheses

We previously detailed the observations made during the event. Since the IPS technology is supposed to record the movements of the participants, the data collected should confirm the observations made. Therefore we use the observations to create a set of hypotheses the data collected have to confirm in order to assess the quality and reliability of ITSI:

1. Talks gather participants closer to the stage
2. Participants naturally regroup by interest over time

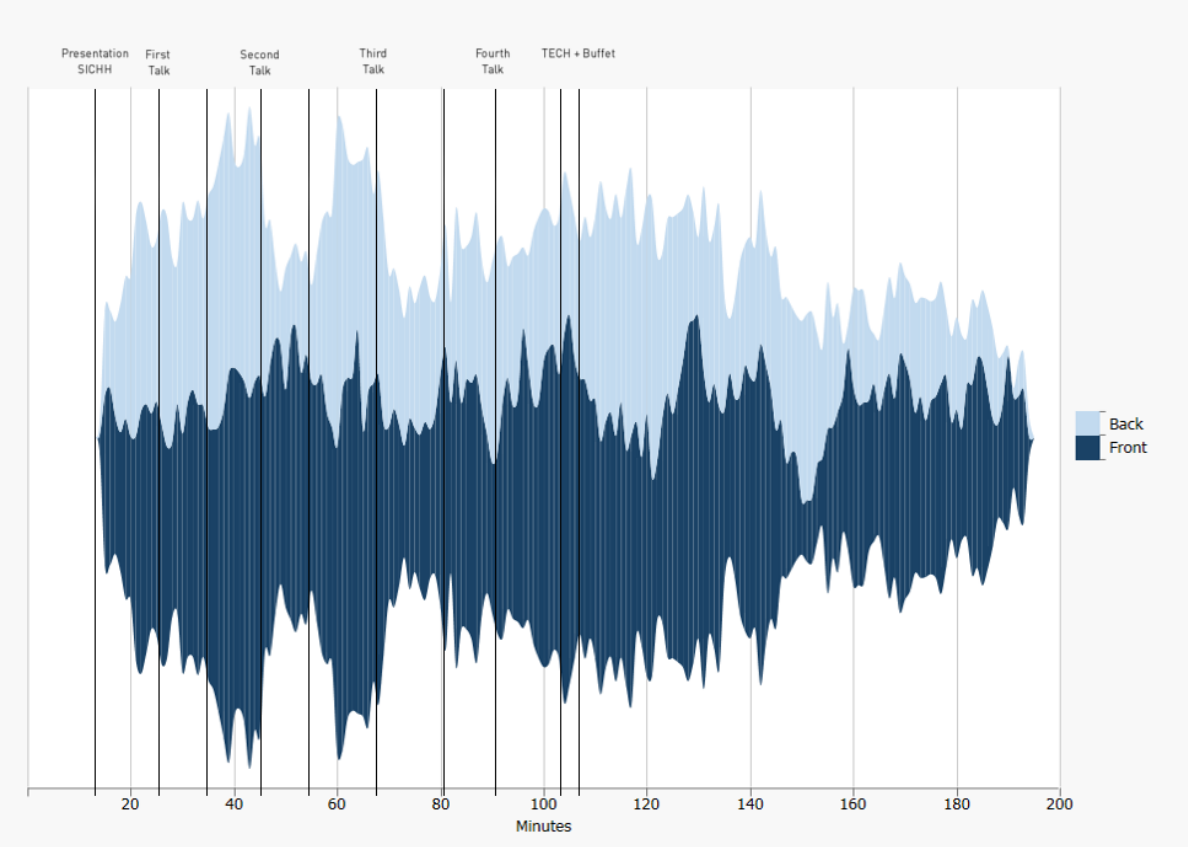
3. Participants resupply on food at the beginning of a networking session
4. The fourth talk captivated the audience more than the others
5. The third talk failed to keep the audience captivated

To prove that ITSI works, we have to prove that each hypothesis can be verified with the data collected. For the first hypothesis we may analyse the evolution over time of two clusters of data: all the records from the front networking tables compared to the ones in the rear of the room. By observing the variations at the beginning of each talk, we will be able to deduce if more participants were indeed moving closer to the stage during talks. For the second hypothesis, comparing the distribution of interests per table for each of the three networking sessions gives an insight on the geographic evolution of interests. As explained in the observations, the live visualization failed to be responsive enough to actually be helpful, hence we will observe a natural evolution. For the third hypothesis, spikes of activity around the buffet tables should be noticed at the beginning of each networking session. Finally, for hypotheses four and five, showing that the participants either increase their time spent at the buffet tables or that more tags are recorded during a talk would be interpreted as a reduced attention from the participants.

7.2.3 Validation: hypothesis 1

The first hypothesis claims that participants move closer to the stage when a talk is given and then reoccupy the room more evenly during the networking sessions. To verify if the data collected by ITSI reflects this observation, we regrouped the count of tags read per seconds from two groups of readers. The "front" group includes all tables from 1 to 6, and the "back" group all the tables from 7 to 10. These two groups of data are represented in figure 13. Additional time markers have been added to identify the beginning of each talk.

Figure 13: Repartition of interests for the first networking session



For each networking session (second marker after a talk), a spike of activity is visible in the "back" group. Also during each talk, the amount of tags read in the front of the room is systematically bigger

than the total of tags read in the back. These variations happen quickly and precisely at the locations of the markers. Therefore ITSI succeeded in reflecting the reality of the event.

7.2.4 Validation: hypothesis 2

The event featured four talks and three distinct networking sessions during which the participants have exchanged information and discussed. In order to verify the second hypothesis — "Participants naturally regroup by interest", we gathered three sets of data. Each set contains the interest distribution per table over a networking session period of time. Each table has a size representing the number of participants who have been detected near the tables over the period of the networking. The numbers by the pie representation are the respective table indexes. By analysing the evolution of gatherings, we observe that the second hypothesis is confirmed by ITSI data.

Figure 14 shows the interest distribution for the first networking session. We can observe that tables 9 and 10 have a dominant interest, but the other tables are mixed. The mixed tables feature either 2 to 3 important interests but none big enough to be considered dominant.

Figure 14: Interests repartition for the first networking session

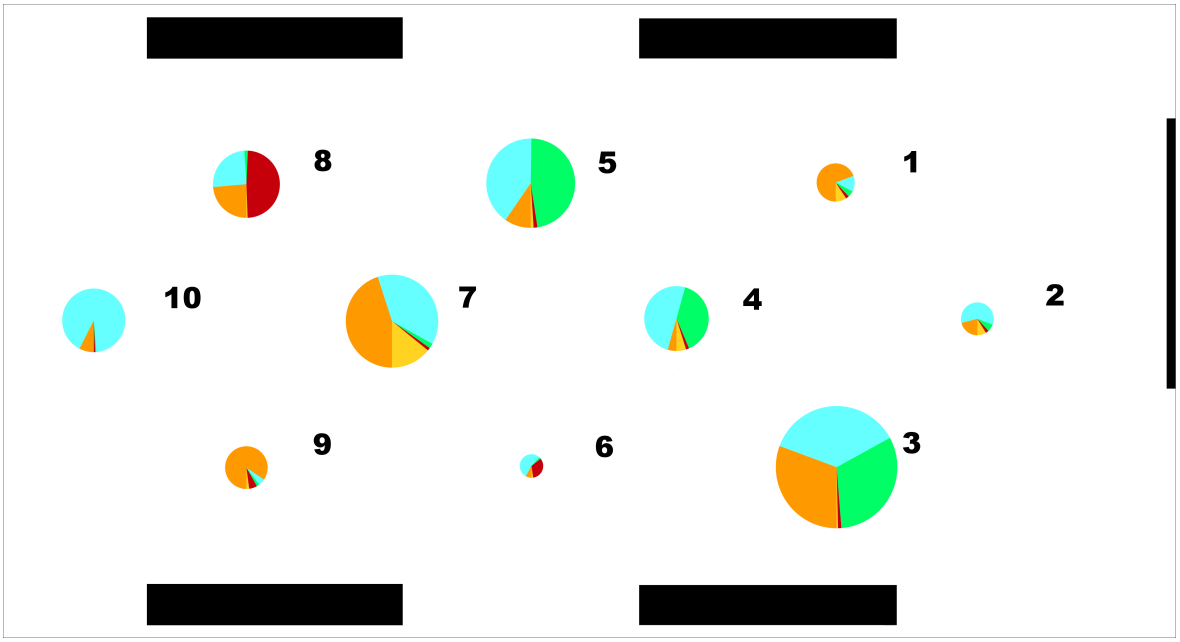


Figure 15 represents interests distribution for the second networking session. When comparing with figure 14, tables 9 and 10 have barely changed in terms of interests distribution. Both tables have reinforced their already dominant theme. On tables 3, 6, 7, and 8 for example one interest has emerged and is becoming dominant. This interest is rarely the one that was already in place during the first networking and we may conclude that participants moved away from certain tables, leaving place for other participants to join. Through this process of mixing, we begin to see tables with a defined interest.

Another noticeable difference is the amount of participants around the tables. Indeed, in the first networking session the room was almost evenly occupied with the exception of table 6 and 2. In the second networking, we observe that the table in the centre of the room have gained attendance, mainly tables 4, 5 and 7 which start to show dominant interests.

Finally, during the third talk we can see that most tables have one dominant interest and a second important one. Over all the third networking session seems to be an extension of the previous one with a bit of movement.

Analysing the geographic evolution of the interests during the three networking sessions shows that the participants have naturally gathered together over time. Each networking session was separated by a talk,

Figure 15: Distribution of interests for the second networking session

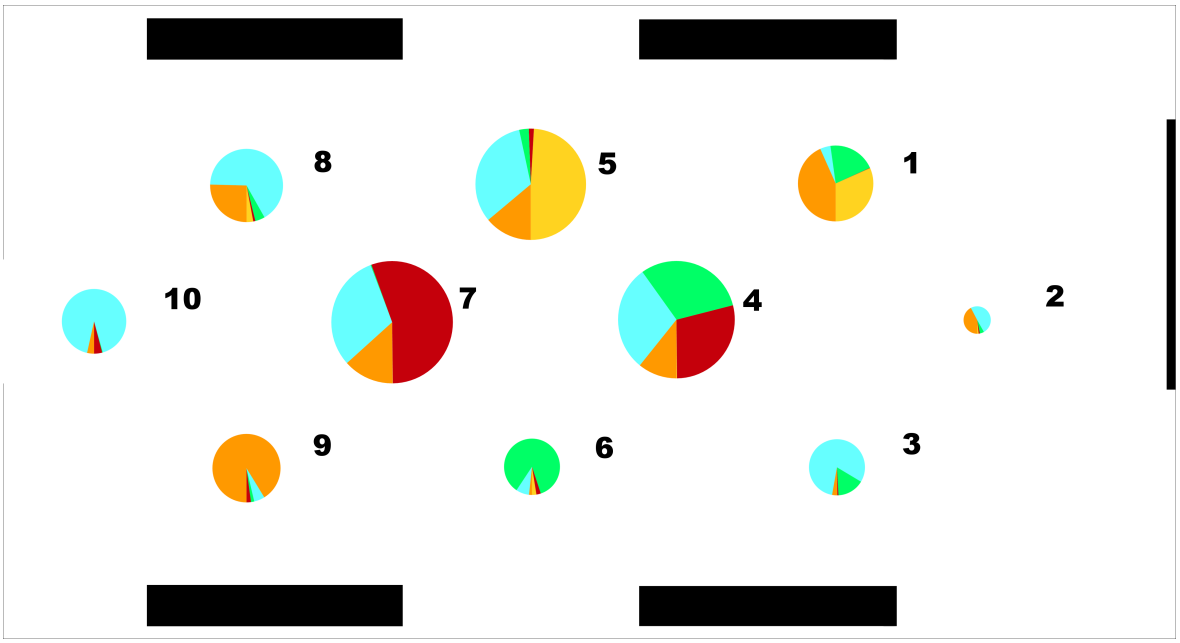
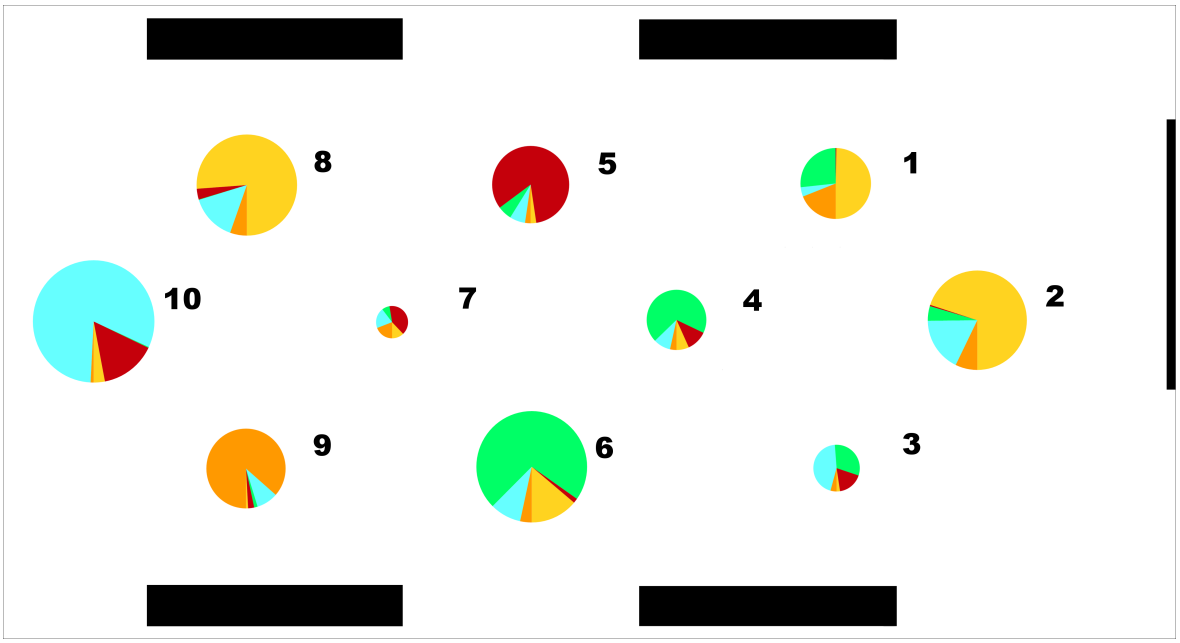


Figure 16: Distribution of interests for the third networking session



which gave opportunities to mix the participants as they moved in the room to get closer to the stage. ITSI successfully recorded the way the participants naturally gathered by interest over time.

7.2.5 Validation: hypothesis 3

With this part of the event, the ITSI fulfils another potential need for research. Indeed, being able to track test subjects, participants in our case, throughout the event allows us to perform data collection passively and more accurately than previously achieved. User evaluations for behavioural tests usually consist of a form the users fill at the end of a test. Unfortunately forms don't always yield results that represent the reality of a test truthfully. The accuracy of data gained through forms tends to be mitigated whether by badly formulated questions or the test subject's answer being unclear and subject to interpretation.

Moreover, it requires additional overhead of work to collect data and store these digitally. In our case we used ITSI to collect all the data. This means the data is truthful to reality and automatically collected without disturbing the test subjects. In this section, we analyse and discuss the data collected during the event for the food experiment.

Figure 17: Stream graph of all four food tables

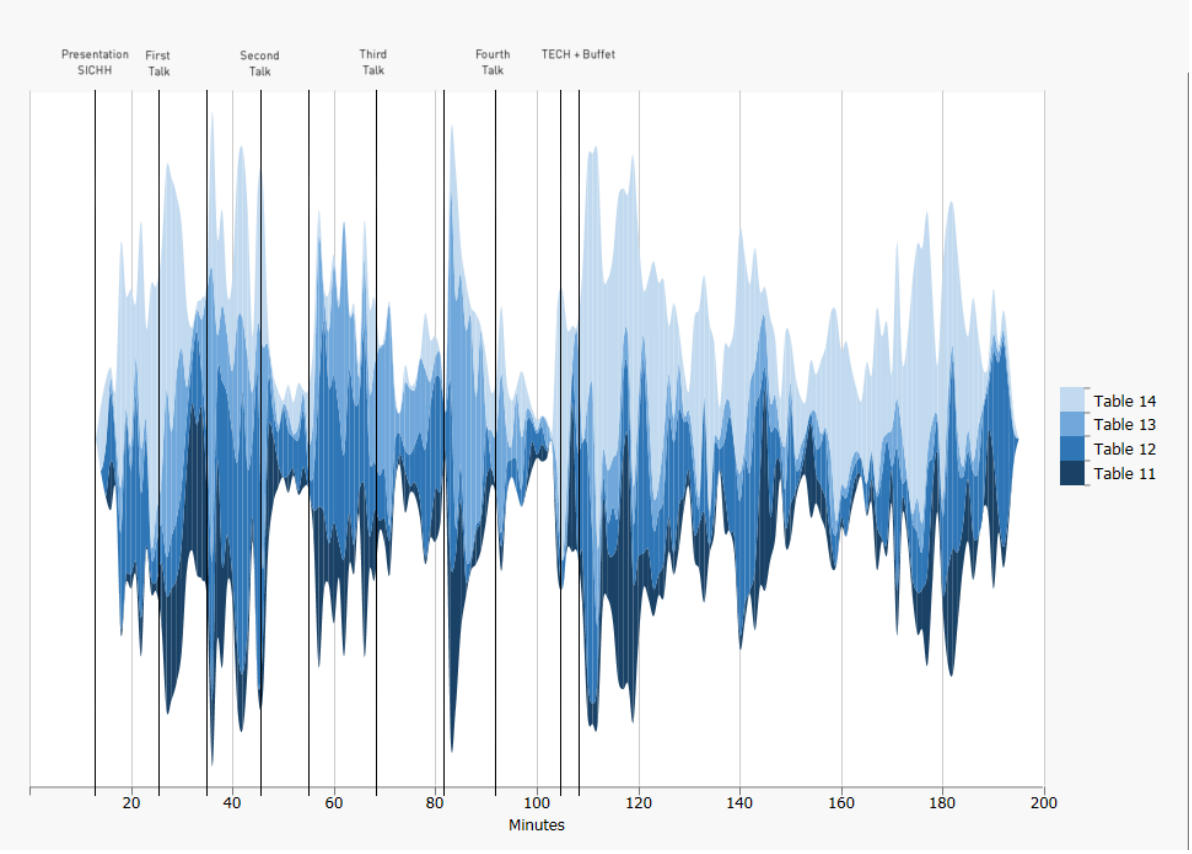


Figure 17 shows the number of tags read per food table over time. It is interesting to notice the participants tend to highly occupy the food table just after a talk has been given. We can notice a spike in tags read at the beginning of a networking session almost each time. We interpret this as the participants quickly refilling their food plate before engaging into networking. We may as well compare the raw popularity of the food tables by their respective total amount of tags read. Tables 12 and 14 are more popular but this result is affected by the fact that the tables are closer to the entrance/exit of the room. Tables 11 and 13 are located in the rear part of the room and got less visited. One can already compare between tables 12-14 and 11-13 that tables 11 and 14 show more popularity overall.

7.2.6 Validation: hypotheses 4 and 5

Another possible observation is the number of tags read during the talks. We may see that during the first talk the participants did resupply on food more than during the other talks. The reasons might be people thinking about securing food and the fact that they just arrived. For the talks 2, 3 and 4 the participants were already resupplied, therefore we may tie the attendance level of food tables with interest level of participants for these talks. We may see talk 2 and 4 present a very low number of readings, with 4 being the lowest as less and less participants went to get food during that talk. We may interpret this as participants paying more attention. However, the third talk shows participants starting to move back to the food tables over time after the first third of the presentation.

7.3 System performances

For the live test of ITSI and the visualizations, we used a Lenovo Thinkpad T440s, equipped with an Intel core I7-4600U — 4MB Cache, up to 3.30GHz. All the readers were connected to this computer through the USB hubs on the three USB 2.0 ports. We computed the overlap between the networking tables to 0% and no more than 1.65% for the buffets tables. The distance and orientation of the tags worn by the participants ensured a tag could be read only from one networking table at a time. In total, ITSI gathered data from 14 RFID readers, reading at a frequency of 1 Hertz. The event received 120 participants — each wearing one passive RFID tag — regrouped under five different interests. The total count of tags read reached 109039 during the 194 minutes of the event.

7.4 Conclusions of experiments

In this chapter we have described observations performed during the event itself. Out of these observations a set of hypotheses has been determined and the real life answers to these are known. By extracting raw data from ITSI's database, statistics have been computed and converted to graphical representations. In each case, we established that ITSI recorded accurate data that verifies all the hypotheses.

8 Conclusions

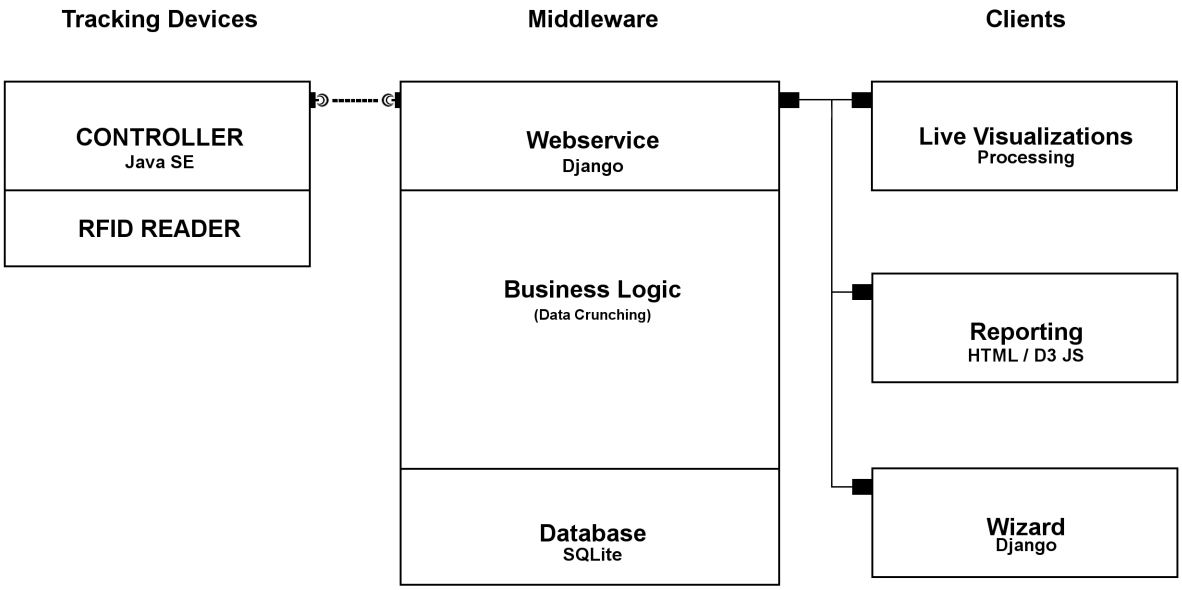
8.1 Wrap up

This project was born out of the idea of augmenting selected social aspects of live events. This task has been divided into two different parts: (1) the need for an indoor positioning system and (2) a layer of data abstraction to make it comprehensive and clear. The indoor positioning system creation required extensive research on the existing technologies, but mostly on the hardware combinations that power these systems. Giants of information technologies such as Google or Nokia have tackled this challenge already and propose solutions relying on the users' smartphone capabilities to locate them. Smartphones are the host of much personal data, and granting access to these can be off-putting to end users. Our approach had to passively track participants of social events without using any or their belonging devices. After analysing the most popular and mature technologies that enable tracking, we chose to use RFID readers combined with passive tags the participants would wear. The RFID technology is used to track merchandise and enhance security around it. A few applications for human tracking have been developed but none of these focus on tracking social interactions.

Throughout this project we created a functional indoor positioning system based on RFID technologies, as depicted in figure 18. Rather than tracking all the participants continuously, we favoured zoning as it focuses only on interest areas where the data collected is likely to be meaningful for later analysis. Finding suitable hardware to come up with a working prototype proved to be a greater challenge than anticipated. The RFID technology being very mature, there is a wide range of products available on the market. We finally settled for USB portable RFID readers that proved handy and suitable, but with two main limitations: (1) the read range proved to vary depending on the position of the tags and the radius was not big enough for all applications, (2) the SDK for this reader model was unfortunately not compatible with Android. We therefore had to find solutions — for the first problem the CSEM managed to attach other antennas to the readers. For the later issue, we had to resign from using Android altogether and focus on a Java-based solution that could reuse parts of the code we already had developed. Nevertheless, the end result is a complete and working indoor positioning system.

The development time frame was constrained by the live event where we had the opportunity to perform stress tests and validation of the data captured with the indoor positioning system in real conditions. To deal with this essential deadline, the whole work has been focused on establishing a first transversal prototype with very limited capabilities, yet fully functional. With the technological setbacks, the first prototype was ready one month before the deadline. Despite its working state, it required further testing and bug-solving until it became reliable enough. Substantial modifications such as adding the sliding window to the live visualization were implemented only one week before the deadline. The live visualization, although working, proved to be lacking in responsiveness with so much data to deal with. The timing

Figure 18: Overview of the final system



management in the web page that hosts the visualization is probably to be held responsible for the major part of this issue as it was set to refresh every 5 seconds. Further tests to render the visualization more responsive are in order. Once the live test was completed, the data has been reviewed and compiled into meaningful information. During the live test, a set of human observations have been made about social behaviour and crowd movements in the room. We used these observations to create a set of hypotheses the data collected by the indoor positioning system had to verify to assess its working condition. All five hypotheses have been verified and further analysis is presented since the system proved to be trustworthy.

In the end, the system proved to be reliable enough but needing better performance in order to be scalable to larger events and more stringent requirements. We also believe that having a tracking system based on a single technology can only lead to a limited accuracy and dataset. Using fusion of various types of sensors is, in our opinion, the next big step for this system. In this project we focused our work on a live visualization and reports to help people with improving their social interactions. Collecting data automatically resulted in a better comprehension of the event and yielded valuable information. Successfully tracking humans in their environments could potentially lead to automated emergency healthcare dispensing systems, security advising systems, guidance of crowds though stores as much as though a building on fire. Potential applications for such a technology are virtually unlimited and yet to be explored.

8.2 Contributions

Through this project, the ITSI IPS has been designed from scratch using RFID technologies. Our research pointed us toward having participants wear a passive tag that would be detected — hence located — near active RFID beacons. The system has been tested in a real environment and it successfully tracks participants' movements and trends. On top of ITSI, a web service with data access and storage capabilities allows the data collected by ITSI to be used by various interfaces. These interfaces are live visualization of events, database management wizard and data reporting graphs. This project has illustrated that social interactions can be tracked in a transparent fashion and yield valuable information to better understand the social interactions during events.

8.3 Future work

8.3.1 Database performance

The database proved to resist to the live event we used to benchmark it. We had 120 participants and 1 client interface running on a single laptop. Using a more powerful database is required in anyway since

SQLite is a development-purposed only database. The database should therefore either be replaced with a MySQL or Postgre SQL database for improved performance. If the needs for this project exceed by far this setup, it would be wise to consider distributing the layers of the system. The database could be transformed into a non-relational database to allow multiple clients to run at the same time. Using a non-relational database does not mean a redesigning the whole database. Django only requires some modifications to the database model to support non-relational databases.

8.3.2 Web service improvements

The web service currently offers a limited set of methods. Although being quite generic, a new set of private methods should be implemented to allow for the creation of data cubes for reporting. A data cube is a subset of data computed from the database. When an event is over, the web service should create a new set of tables from the event and crunch data. This would grant easy access to data that would otherwise need computations on request. Most of the statistical methods take a long time to complete and recomputing these on the fly for each request is an inefficient use of processing-power and energy.

8.3.3 Tracking improvements

From the live event we used as a benchmark opportunity, we noticed that the tags sometimes had difficulties to be read by the readers. The main reason is the improper orientation of the tags. Many participants had the tags facing slightly up when attached at the level of their chest. Therefore the tags were not facing the readers, which lowered signal gain and reduced the read range. A solution would be to design holders for the RFID tags that already orient the tags a bit toward the floor. Another way would be to ask the participants to lay down their tags on the table while they discuss there, but the inconvenience caused by this would tamper with the natural behaviour of the participants. Another lead would be to try to manually setup the RFID readers to favour read range over read speed. There is an option in the SDK provided with the RFID readers that allow to specify a string of parameters, including one for read range optimization.

8.3.4 Visualization improvements

The visualization proved to be lacking in terms of responsiveness and refresh rate. Therefore an improvement would be to modify the way the data is sent to the visualization. The web service could use multicast and the classic publish/subscribe system to send the event information to the clients. A client such as a live visualization would have to register to an event and as soon as this event receives data from the readers controllers, the data would be sent to all the clients who subscribed to the event. This would remove the constraint of having a constant frequency of rendering. Moreover each table could be a stream of data, and each table rendered individually. With asynchronous update of the tables, the visualization can be more efficient by only process changes.

Another part of the visualization that would become convenient is a graphical event editor. Currently using the Django administration panel works but lacks intuitiveness in practice. Moreover it requires the user to be familiar with the database model. Therefore it would be relevant to have a graphical editor with a step-by-step wizard that would allow to do the following:

1. Show a form to create the event
2. Show a graphical editor to assign a location to each sensor
3. Show a form to add all the participants with their respective interest tag, and automatically bind them to the selected event.

These three operations are currently time-consuming and not intuitive, hence the need for an easy and documented step-by-step wizard. For the reporting visualizations, having an automated board that allows the user to retrieve pre-computed data and generate graphics and statistics out of these would be a plus. Currently the statistics currently have to be extracted from the database using the stock methods we implemented, then filtered and manually turned into graphics. Finally, for client interfaces that require good performances and interaction with the data, using more powerful engines such as game engines could

lead to a much smoother and reactive experience. By using an engine like Unity 5, it is possible to create a project that exploits the data from the web service and grants much more freedom of visualization by supporting more features.

8.3.5 Hardware and security

During our first development iteration, we planned on using Android tablets as reader controllers. They indeed offered a screen to perform WYSIWYG configuration operations and they are also able to power the readers. The most important advantage is the simplicity of installation since every aspect works wirelessly. However the SDK of the RFID readers, although advertised as compatible with Android, is actually only compatible with the newest Mercury 6e processing units. Since the SDK is available for many development platforms and programming languages, it would be interesting to explore the use of an Arduino boadr combined with USB and Wi-Fi shields since it could replace Android tablets. In that case, the transmissions should be SSL encrypted.

Another security-related point is the cross-domain restrictions to guarantee security. We have currently disabled the Cross-Site Request Forgery (CSRF) management in Django to better focus on the web service methods. CSRF is also known as "Confused Deputy" attack or "Session Riding" [7]. A session can be hacked by sending information to a server using the session information of another user, thus high-jacking this user. CSRF can be used to prevent such attacks by using a token management system. A token is exchanged between the server and the client to add an extra layer of authentication information for future requests. It would be wise to fully enable this functionality in the web service.

8.3.6 Additional tests

As aforementioned, the live visualization did not yield the expected results because of its refresh rate. The participants quickly lost interest in the technology as the response time was not prompt enough to be truly interactive. Improving the refresh rate and perform new tests would confirm whether ITSI combined with a live visualization improves the gathering process of participants during a social event. To achieve a better refresh rate, the settings in the live visualization should be modified. The refresh rate is currently handled by JavaScript code. Reducing the wait time between two redraws of the visualization would on its own already improve the refresh rate considerably, at the expense of the database server which will have to process more requests. Another way would be to deal with each table asynchronously. In this case, all the tables can feature different refresh rates that could be based on the attendance level at each table. Finally, using a game engine with the sole purpose of rendering the live visualization would be more efficient than a web-based engine. In this case, we would consider Unity 5 to provide a solid dedicated application.

A new batch of tests should be performed once the refresh rate of the live visualization is fixed. This time, a user evaluation aiming at determining the usefulness of the technology in a live context should be performed. We have already shown that the offline data collected by ITSI is accurate and valuable, but proving whether a participant using the live visualization is indeed more efficient and successful in his/her social interactions or not remains to be evaluated.

References

[1] Martin Atzmueller, Dominik Benz, Stephan Doerfel, Andreas Hotho, Robert Jäschke, Bjoern Elmar Macek, Folke Mitzlaff, Christoph Scholz, and Gerd Stumme. Enhancing social interactions at conferences. *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(3):101–107, 2011.

[2] Kenneth C Cheung, Stephen S Intille, and Kent Larson. An inexpensive bluetooth-based indoor positioning hack. *Proc. UbiComp06 Extended Abstracts*, 2006.

[3] Donna Cox, Volodymyr Kindratenko, and David Pointer. Intellibadgetm: Towards providing location-aware value-added services at academic conferences. In *UbiComp 2003: Ubiquitous Computing*, pages 264–280. Springer, 2003.

[4] Michael AH Fried, Anna Fensel, Federico Michele Facca, and Dieter Fensel. An extensible system for enhancing social conference experience. In *Perspectives of Systems Informatics*, pages 95–110. Springer, 2012.

[5] Daniel Gatica-Perez. Automatic nonverbal analysis of social interaction in small groups: A review. *Image and Vision Computing*, 27(12):1775–1787, 2009.

[6] Elisabeth Ilie-Zudor, Zsolt Kemeny, Peter Egri, and Laszlo Monostori. The rfid technology and its current applications. *ISBN*, 963(86586):5, 2006.

[7] Konstantin Käfer. Cross site request forgery, 2008.

[8] Shin'ichi Konomi, Sozo Inoue, Takashi Kobayashi, Masashi Tsuchida, and Masaru Kitsuregawa. Supporting colocated interactions using rfid and social network displays. *Pervasive Computing, IEEE*, 5(3):48–56, 2006.

[9] Yutaka Masumoto. Global positioning system, May 11 1993. US Patent 5,210,540.

[10] Rainer Mautz. *Indoor positioning technologies*. PhD thesis, Habil. ETH Zürich, 2012, 2012.

[11] Joseph F McCarthy, David W McDonald, Suzanne Soroczak, David H Nguyen, and Al M Rashid. Augmenting the social space of an academic conference. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 39–48. ACM, 2004.

[12] Robert Watson-Watt. Radar in war and in peace. *Nature*, 156:319–324, 1945.

[13] Zhengyou Zhang. Microsoft kinect sensor and its effect. *MultiMedia, IEEE*, 19(2):4–10, 2012.

9 Appendices

For the sake of clarity throughout the report, we chose to separate from the report most code listings as well as extra information that would mostly be of interest to developers rather than the regular reader. In the following chapters, more elements of the implementations as well as extra reports coming from the data collected from the live tests are presented and briefly explained.

9.1 Appendix I - Django database model

Listing 1 shows the Django code that defines the objects for the database model. Each table is represented by a class, and each class can host functions declared as "def". We added custom string functions for data clarity, functions to automatically format the data to the JSON standard as well as meta. The meta is used to specify additional options such as which field is used to order the data from a table.

Listing 1: Database - Object model

```
from django.db import models

# DB Color/Interest tags field.
class InterestTag(models.Model):
  description = models.CharField(max_length=100)
  color = models.CharField(max_length=7)

  def __str__(self):
    return self.description

  def as_json(self):
    return dict(
      description = str(self.description),
      color = str(self.color))

  class Meta:
    ordering = ('description',)

# DB Participant fields.
class Participant(models.Model):
  reference = models.IntegerField(unique=True, blank=True, null=True)
  tagId = models.CharField(max_length=255)
  sex = models.CharField(max_length=1, blank=True, null=True)
  interestTag = models.ForeignKey(InterestTag, blank=True, null=True)

  def __str__(self):
    return str(self.reference) + ' | ' + self.tagId

  def as_json(self):
    return dict(
      tagID = str(self.tagId),
      sex = str(self.sex),
      interestTag = self.interestTag.as_json())

  class Meta:
    ordering = ('tagId',)

# DB Sensor fields.
class Sensor(models.Model):
  identifier = models.IntegerField(unique=True)
  description = models.CharField(max_length=200)
  x = models.FloatField()
  y = models.FloatField()
  rssiThreshold = models.FloatField()
  radius = models.FloatField()
  displayable = models.BooleanField(default=True)

  def __str__(self):
    return "Sensor: " + self.description

  def as_json(self):
    return dict(
```

```
53        identifier = self.identifier,
54        description = str(self.description),
55        x = self.x,
56        y = self.y,
57        rssiThreshold = self.rssiThreshold,
58        radius = self.radius)
59
60    class Meta:
61      ordering = ('description', 'identifier', )
62
63  # DB Event fields.
64  class Event(models.Model):
65    name = models.CharField(max_length=200)
66    startDate = models.DateTimeField('start date')
67    stopDate = models.DateTimeField('stop date')
68    interestTags = models.ManyToManyField(InterestTag, blank=True, null=True)
69    participants = models.ManyToManyField(Participant, blank=True, null=True)
70    sensors = models.ManyToManyField(Sensor, blank=True, null=True)
71
72    def __str__(self):
73      return self.name
74
75    class Meta:
76      ordering = ('name',)
77
78  # Temporal markers.
79  class Marker(models.Model):
80    label = models.CharField(max_length=200)
81    timeStamp = models.DateTimeField('timeStamp')
82    event = models.ForeignKey(Event)
83
84    def __str__(self):
85      return "Marker: " + self.label
86
87    def as_json(self):
88      return dict(
89        timeStamp = str(self.timeStamp.replace(second=0, microsecond=0))[11:-6],
90        label = str(self.label)
91      )
92
93    class Meta:
94      ordering = ('timeStamp', 'label', )
95
96  class Record(models.Model):
97    tagId = models.CharField(max_length=255)
98    timeStamp = models.DateTimeField('time stamp', null=True)
99    rssi = models.FloatField()
100   event = models.ForeignKey(Event, null=True)
101   sensor = models.ForeignKey(Sensor, null=True)
102
103   def __str__(self):
104     return self.tagId
105
106   def as_json(self):
107     return dict(
108       eventID = self.event.pk,
109       sensorID = self.sensor.identifier,
110       timeStamp = str(self.timeStamp),
111       tagID = str(self.tagId),
112       rssi = self.rssi)
113
114   class Meta:
115     ordering = ('tagId',)
```

9.2 Appendix II - Example of generated statistics output

An example output by the webservice when the requesting the data statistics of event 3, sensor with identifier 1. Time represents the minutes delta since the start of the event, the value represents the number of tags read during a specific minute of the event.

Listing 2: Generated statistics output example


```
[{"time": 0, "value": 0}, {"time": 1, "value": 0}, {"time": 2, "value": 0},
 {"time": 3, "value": 0}, {"time": 4, "value": 0}, {"time": 5, "value": 0},
 {"time": 6, "value": 0}, {"time": 7, "value": 0}, {"time": 8, "value": 0},
 {"time": 9, "value": 0}, {"time": 10, "value": 0}, {"time": 11, "value": 0},
 {"time": 12, "value": 0}, {"time": 13, "value": 0}, {"time": 14, "value": 7},
 {"time": 15, "value": 63}, {"time": 16, "value": 62}, {"time": 17, "value": 65},
 {"time": 18, "value": 65}, {"time": 19, "value": 64}, {"time": 20, "value": 66},
 {"time": 21, "value": 59}, {"time": 22, "value": 65}, {"time": 23, "value": 67},
 {"time": 24, "value": 34}, {"time": 25, "value": 58}, {"time": 26, "value": 57},
 {"time": 27, "value": 57}, {"time": 28, "value": 65}, {"time": 29, "value": 61},
 {"time": 30, "value": 56}, {"time": 31, "value": 54}, {"time": 32, "value": 50},
 {"time": 33, "value": 6}, {"time": 34, "value": 6}, {"time": 35, "value": 3},
 {"time": 36, "value": 14}, {"time": 37, "value": 77}, {"time": 38, "value": 19},
 {"time": 39, "value": 60}, {"time": 40, "value": 50}, {"time": 41, "value": 55},
 {"time": 42, "value": 57}, {"time": 43, "value": 43}, {"time": 44, "value": 31},
 {"time": 45, "value": 79}, {"time": 46, "value": 55}, {"time": 47, "value": 65},
 {"time": 48, "value": 68}, {"time": 49, "value": 59}, {"time": 50, "value": 67},
 {"time": 51, "value": 72}, {"time": 52, "value": 72}, {"time": 53, "value": 67},
 {"time": 54, "value": 66}, {"time": 55, "value": 70}, {"time": 56, "value": 68},
 {"time": 57, "value": 86}, {"time": 58, "value": 43}, {"time": 59, "value": 79},
 {"time": 60, "value": 100}, {"time": 61, "value": 112}, {"time": 62, "value": 86},
 {"time": 63, "value": 87}, {"time": 64, "value": 98}, {"time": 65, "value": 61},
 {"time": 66, "value": 25}, {"time": 67, "value": 44}, {"time": 68, "value": 49},
 {"time": 69, "value": 50}, {"time": 70, "value": 3}, {"time": 71, "value": 8},
 {"time": 72, "value": 7}, {"time": 73, "value": 25}, {"time": 74, "value": 55},
 {"time": 75, "value": 75}, {"time": 76, "value": 71}, {"time": 77, "value": 63},
 {"time": 78, "value": 13}, {"time": 79, "value": 19}, {"time": 80, "value": 27},
 {"time": 81, "value": 20}, {"time": 82, "value": 25}, {"time": 83, "value": 68},
 {"time": 84, "value": 44}, {"time": 85, "value": 74}, {"time": 86, "value": 53},
 {"time": 87, "value": 69}, {"time": 88, "value": 25}, {"time": 89, "value": 20},
 {"time": 90, "value": 29}, {"time": 91, "value": 59}, {"time": 92, "value": 51},
 {"time": 93, "value": 61}, {"time": 94, "value": 80}, {"time": 95, "value": 88},
 {"time": 96, "value": 90}, {"time": 97, "value": 83}, {"time": 98, "value": 73},
 {"time": 99, "value": 87}, {"time": 100, "value": 100}, {"time": 101, "value": 93},
 {"time": 102, "value": 99}, {"time": 103, "value": 109}, {"time": 104, "value": 105},
 {"time": 105, "value": 95}, {"time": 106, "value": 75}, {"time": 107, "value": 72},
 {"time": 108, "value": 61}, {"time": 109, "value": 81}, {"time": 110, "value": 120},
 {"time": 111, "value": 92}, {"time": 112, "value": 74}, {"time": 113, "value": 61},
 {"time": 114, "value": 47}, {"time": 115, "value": 102}, {"time": 116, "value": 93},
 {"time": 117, "value": 99}, {"time": 118, "value": 66}, {"time": 119, "value": 48},
 {"time": 120, "value": 51}, {"time": 121, "value": 34}, {"time": 122, "value": 23},
 {"time": 123, "value": 45}, {"time": 124, "value": 72}, {"time": 125, "value": 90},
 {"time": 126, "value": 45}, {"time": 127, "value": 78}, {"time": 128, "value": 112},
 {"time": 129, "value": 132}, {"time": 130, "value": 128}, {"time": 131, "value": 120},
 {"time": 132, "value": 95}, {"time": 133, "value": 104}, {"time": 134, "value": 133},
 {"time": 135, "value": 67}, {"time": 136, "value": 82}, {"time": 137, "value": 86},
 {"time": 138, "value": 111}, {"time": 139, "value": 125}, {"time": 140, "value": 106},
 {"time": 141, "value": 56}, {"time": 142, "value": 85}, {"time": 143, "value": 77},
 {"time": 144, "value": 7}, {"time": 145, "value": 38}, {"time": 146, "value": 25},
 {"time": 147, "value": 2}, {"time": 148, "value": 2}, {"time": 149, "value": 7},
 {"time": 150, "value": 3}, {"time": 151, "value": 1}, {"time": 152, "value": 6},
 {"time": 153, "value": 0}, {"time": 154, "value": 1}, {"time": 155, "value": 16},
 {"time": 156, "value": 25}, {"time": 157, "value": 82}, {"time": 158, "value": 54},
 {"time": 159, "value": 77}, {"time": 160, "value": 68}, {"time": 161, "value": 88},
 {"time": 162, "value": 76}, {"time": 163, "value": 56}, {"time": 164, "value": 45},
 {"time": 165, "value": 68}, {"time": 166, "value": 95}, {"time": 167, "value": 98},
 {"time": 168, "value": 103}, {"time": 169, "value": 112}, {"time": 170, "value": 106},
 {"time": 171, "value": 117}, {"time": 172, "value": 135}, {"time": 173, "value": 163},
 {"time": 174, "value": 114}, {"time": 175, "value": 103}, {"time": 176, "value": 107},
 {"time": 177, "value": 96}, {"time": 178, "value": 150}, {"time": 179, "value": 113},
 {"time": 180, "value": 113}, {"time": 181, "value": 89}, {"time": 182, "value": 122},
 {"time": 183, "value": 144}, {"time": 184, "value": 133}, {"time": 185, "value": 146},
 {"time": 186, "value": 139}, {"time": 187, "value": 113}, {"time": 188, "value": 102},
```

64 {"time": 189, "value": 120}, {"time": 190, "value": 48}]

9.3 Appendix III - Activity graphs for the SICHH Forum event

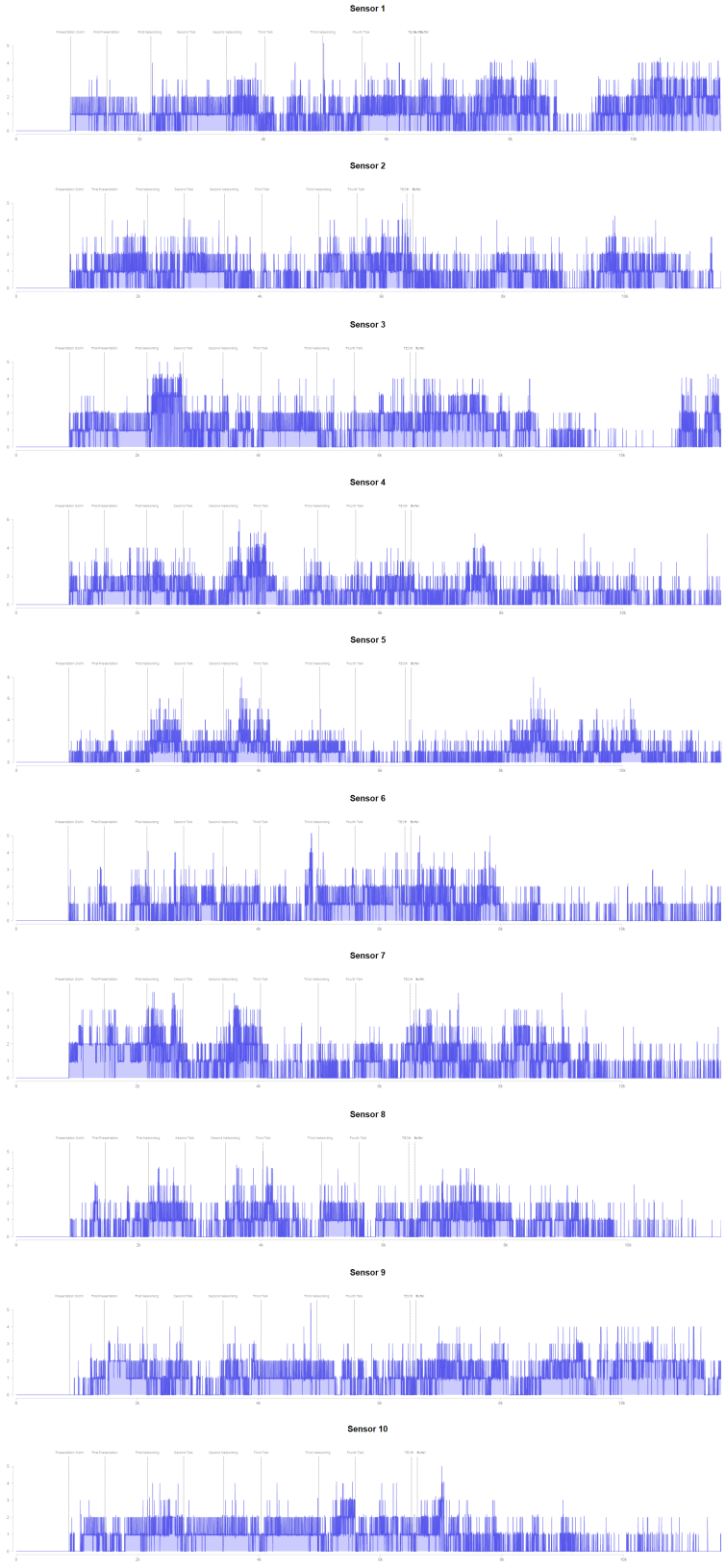


Figure 19: Statistics of tags read per second for networking tables

9.4 Appendix IV - Activity graphs for the SICHH Forum experiment

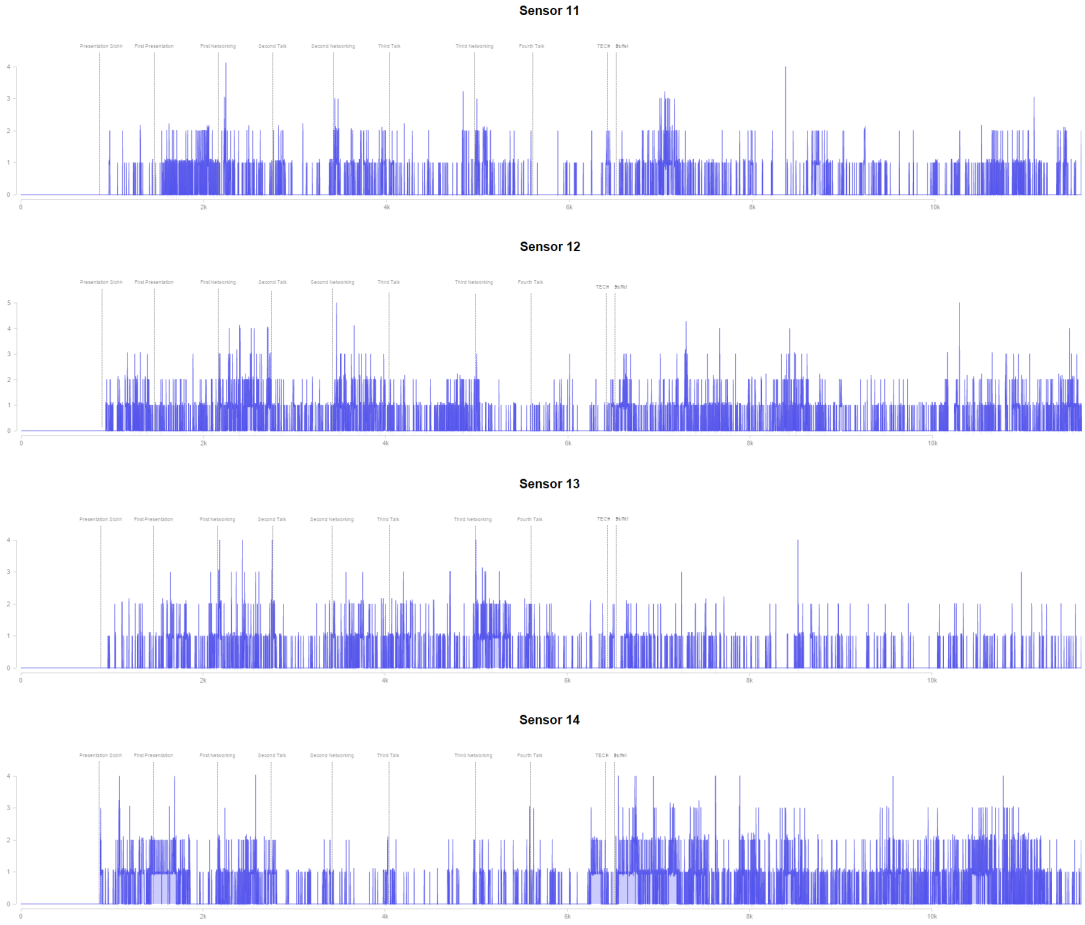


Figure 20: Statistics of tags read per second for food tables

9.5 Appendix V - Web service methods

The following code listings present the code used in Django to create the views. Each view can be called via an URL and perform a set of computation.

Listing 3: Web Service - Record input

```
# Post a new record for an event & sensor.
@csrf_exempt
def input_record(request):
  if request.method == 'POST':
    try:
      # Read data from the JSON payload.
      raw = yaml.load(request.body)
      str_data = str(raw).replace("'", '"')
      data = json.loads(str_data)

      i_eventID = data["eventID"]
      i_sensorID = data["sensorID"]
      i_timeStamp = data["timeStamp"]
      i_records = data["records"]

      # Convert time stamps correctly.
      c_timestamp = datetime.datetime.strptime(str(i_timeStamp), "%Y-%m-%d %H:%M:%S")
      # Acquire relations in database.
      e = Event.objects.get(pk=i_eventID)
      s = Sensor.objects.get(identifier=i_sensorID)

      # Write the records.
      for rec in i_records:
        Record.objects.create(event=e, sensor=s, timeStamp=c_timestamp, tagId=rec["tag"],
            rssi=rec["rssi"])

    except:
      logging.debug('record failed')
      return HttpResponse('FAILED')

  return HttpResponse("OK")
```

Note: the decorator "@csrf_exempt" disables the cross-domain validation HTTP imposes. For prototyping purposes, we did not judge useful to implement certificates to validate an input. Moreover, all communications are done through Wi-Fi in a closed network.

Listing 4: Web Service - Statistics marker input

```
# Post a new marker.
@csrf_exempt
def input_marker(request, eventID, label):
  if request.method == 'POST':
    try:
      i_eventID = int(str(unicode(eventID)))
      i_label = str(unicode(label))
      c_timestamp = datetime.datetime.now()

      # Acquire relations in database.
      e = Event.objects.get(pk=i_eventID)

      # Add new marker.
      Marker.objects.create(label=i_label, timeStamp=c_timestamp, event=e)

    except:
      logging.debug('Posting marker failed')
      return HttpResponse('FAILED')

  return HttpResponse("OK")
```

Listing 5: Web Service - Server Time

```
# Get the time of the webservice.
def time(request, diff):
  if 'callback' in request.REQUEST:
    # Send a JSONP response.
    data = '%s({\'timeStamp\' : \'%s\'});' % (request.REQUEST['callback'], (datetime.
        datetime.now() - datetime.timedelta(seconds=int(diff))).replace(microsecond=0))
    return HttpResponse(data, "text/javascript")

  return HttpResponse(str((datetime.datetime.now() - datetime.timedelta(seconds=int(diff))).
      replace(microsecond=0)))
```

Listing 6: Web Service - Live Update

```
# Get updated live feed.
@csrf_exempt
def output_getLiveUpdate(request, eventID, timeStamp):
  if request.method == 'GET':
    try:
      i_eventID = int(str(unicode(eventID)))
      i_timeStamp = str(unicode(timeStamp))

      # Prepare request context.
      c_timestamp = datetime.datetime.strptime(i_timeStamp, "%Y-%m-%d %H:%M:%S")

      # Gather list of records.
      records = Record.objects.filter(event__pk=i_eventID).filter(timeStamp=c_timestamp).
          exclude(sensor__displayable=False)

      # Sort distinct results.
      distinctRecords = []
      seen = set()

      for rec in records:
        if rec.tagId not in seen:
          distinctRecords.append(rec)
          seen.add(rec.tagId)

      results = [ob.as_json() for ob in distinctRecords]
      print results

      if 'callback' in request.REQUEST:
                # Send a JSONP response.
        data = '%s(%s);' % (request.REQUEST['callback'], results)
        return HttpResponse(data, "text/javascript")

    except:
      logging.debug('request record failed')
      return HttpResponse('FAILED')

    return HttpResponse(json.dumps(results), "application/json")
```

Listing 7: Web Service - Output configuration

```
# Get config file.
def output_config(request, eventID):
  if request.method == 'GET':
    try:
      sensors = Sensor.objects.filter(event__pk = eventID).exclude(displayable=False)
      results = [ob.as_json() for ob in sensors]

      if 'callback' in request.REQUEST:
        # Send a JSONP response.
        data = '%s(%s);' % (request.REQUEST['callback'], results)
        return HttpResponse(data, "text/javascript")

    except:
      logging.debug('request config failed')
```

```
15        return HttpResponse('FAILED')
16
17      return HttpResponse(json.dumps(results), "application/json")
```

Listing 8: Web Service - Output all records

```
1  # Get All records for an event.
2  def output_all_records(request, eventID):
3    if request.method == 'GET':
4      try:
5        records = Record.objects.filter(event__pk=eventID)
6        results = [ob.as_json() for ob in records]
7
8      except:
9        logging.debug('request all records failed')
10       return HttpResponse('FAILED')
11
12   return HttpResponse(json.dumps(results), "application/json")
```

Listing 9: Web Service - Output all interest tags

```
1  # Get All records for an event.
2  def output_all_interestTags(request, eventID):
3    if request.method == 'GET':
4      try:
5        tags = InterestTag.objects.filter(event__pk=eventID)
6        results = [ob.as_json() for ob in tags]
7
8      except:
9        logging.debug('request all interest tags failed')
10       return HttpResponse('FAILED')
11
12   return HttpResponse(json.dumps(results), "application/json")
```

Listing 10: Web Service - Output all participants

```
1  // URL.
2  # Get All records for an event.
3  def output_all_participants(request, eventID):
4    if request.method == 'GET':
5      try:
6        participants = Participant.objects.filter(event__pk=eventID)
7        results = [ob.as_json() for ob in participants]
8
9        if 'callback' in request.REQUEST:
10                 # Send a JSONP response.
11         data = '%s(%s);' % (request.REQUEST['callback'], results)
12         return HttpResponse(data, "text/javascript")
13
14     except:
15       logging.debug('request all interest tags failed')
16       return HttpResponse('FAILED')
17
18   return HttpResponse(json.dumps(results), "application/json")
```

Listing 11: Web Service - Output all time markers

```
1  # Get all the markers for a given event.
2  def output_statMarkers(request, eventID):
3    if request.method == 'GET':
4      i_eventID = int(str(unicode(eventID)))
5
6      # Get starting time of the event to compute the time difference.
```

```
7       eventObject = Event.objects.filter(pk=i_eventID)
8       eventTime = eventObject[0].startDate
9
10      # Gather list of markers.
11      markers = Marker.objects.filter(event__pk=i_eventID)
12      count = Marker.objects.filter(event__pk=i_eventID).count()
13      results = '['
14
15      if(count > 0):
16        for x in range(0, count):
17          # Compute the time difference.
18          diff = ((markers[x].timeStamp - eventTime).seconds) +1
19
20          # Add the data to the json response.
21          if(x != count -1):
22            results += '{"time": ' + str(diff) + ', "label": "' + markers[x].label + '"}, '
23          else:
24            results += '{"time": ' + str(diff) + ', "label": "' + markers[x].label + '"}]'
25      else:
26        results = '[]'
27
28      if 'callback' in request.REQUEST:
29          # Send a JSONP response.
30          data = '%s(%s);' % (request.REQUEST['callback'], results)
31          return HttpResponse(data, "text/javascript")
32
33    return HttpResponse(json.dumps(results), "application/json")
```

Listing 12: Web Service - Raw statistical data

```
1  # Get all records for a given event and sensor.
2  def output_statData(request, eventID, sensorID):
3    if request.method == 'GET':
4      try:
5        # Gather required data.
6        i_eventID = int(str(unicode(eventID)))
7        i_sensorID = int(str(unicode(sensorID)))
8
9        eventObject = Event.objects.filter(pk=i_eventID)
10       eventTime = eventObject[0].startDate
11
12       records = Record.objects.filter(event__pk=i_eventID).filter(sensor__identifier=
             i_sensorID).order_by('timeStamp')
13       count = Record.objects.filter(event__pk=i_eventID).filter(sensor__identifier=
             i_sensorID).count()
14
15       # Determine the number of minutes of the records.
16       if(count > 1):
17         tdelta = records[count-1].timeStamp - records[0].timeStamp
18         secondsTotal = (tdelta.seconds) +1
19         statistics = [0] * secondsTotal
20
21         # Crunch the data.
22         for x in range(0, count):
23           index = ((records[x].timeStamp - records[0].timeStamp).seconds)
24           statistics[index] += 1;
25
26         # Format the values to json.
27         results = '['
28
29         secondsCount = 0
30         startTime = records[0].timeStamp
31         eventTmpTime = eventTime
32         timeDiffEventBegin = (startTime - eventTmpTime).seconds
33
34         for z in range(0, timeDiffEventBegin):
35           results += '{"time": ' + str(secondsCount) + ', "value": 0}, '
36           secondsCount += 1
37
38         for y in range(0, secondsTotal):
```

```
39            # Add the data to the json response.
40            if(y != secondsTotal -1):
41              results += '{"time": ' + str(secondsCount) + ', "value": ' + str(statistics[y])
                    + '}, '
42              secondsCount += 1
43            else:
44              results += '{"time": ' + str(secondsCount) + ', "value": ' + str(statistics[y])
                    + '}]'
45
46          if 'callback' in request.REQUEST:
47            # Send a JSONP response.
48            data = '%s(%s);' % (request.REQUEST['callback'], results)
49            return HttpResponse(data, "text/javascript")
50
51        else:
52          return HttpResponse('NONE')
53
54      except:
55        logging.debug('request record failed')
56        return HttpResponse('FAILED')
57
58  return HttpResponse(json.dumps(results), "application/json")
```

Listing 13: Web Service - Interest-centered statistical data

```
1  # Output statistics per interest tag for a sensor.
2  def output_statData_perTag(request, eventID, sensorID):
3    if request.method == 'GET':
4      try:
5        # Gather required data.
6        i_eventID = int(str(unicode(eventID)))
7        i_sensorID = int(str(unicode(sensorID)))
8
9        eventObject = Event.objects.filter(pk=i_eventID)
10
11       interestTags = InterestTag.objects.filter(event__pk=i_eventID).order_by('id')
12       countInterestTags = InterestTag.objects.filter(event__pk=i_eventID).count()
13       interestsDict = {}
14
15       eventTime = eventObject[0].startDate
16
17       records = Record.objects.filter(event__pk=i_eventID).filter(sensor__identifier=
             i_sensorID).order_by('timeStamp')
18       count = Record.objects.filter(event__pk=i_eventID).filter(sensor__identifier=
             i_sensorID).count()
19
20       # Create interest tags hashmap.
21       for tagsTmpCount in range(0, countInterestTags):
22         interestsDict[interestTags[tagsTmpCount].id] = tagsTmpCount
23
24       # Determine the number of minutes of the records.
25       if(count > 1):
26         tdelta = records[count-1].timeStamp - records[0].timeStamp
27         secondsTotal = (tdelta.seconds) +1
28         statistics = np.zeros((countInterestTags, secondsTotal))
29
30         # Crunch the data.
31         for x in range(0, count):
32           index = ((records[x].timeStamp - records[0].timeStamp).seconds)
33
34           if(Participant.objects.filter(tagId=str(records[x].tagId)).count() > 0):
35             tag = interestsDict[Participant.objects.filter(tagId=str(records[x].tagId))[0].
                   interestTag.id]
36             statistics[tag][index] += 1;
37
38         # Format the values to json.
39         results = '['
40
41         for t in range(0, countInterestTags):
42           # Compute array per interest tag.
```

```
43            array = '['
44            secondsCount = 0
45            startTime = records[0].timeStamp
46            eventTmpTime = eventTime
47            timeDiffEventBegin = (startTime - eventTmpTime).seconds
48
49            for z in range(0, timeDiffEventBegin):
50              array += '{"x": ' + str(secondsCount) + ', "y": 0}, '
51              secondsCount += 1
52
53            for y in range(0, secondsTotal):
54              # Add the data to the json response.
55              if(y != secondsTotal -1):
56                array += '{"x": ' + str(secondsCount) + ', "y": ' + str(statistics[t][y]) + '
                      }, '
57                secondsCount += 1
58              else:
59                array += '{"x": ' + str(secondsCount) + ', "y": ' + str(statistics[t][y]) + '
                      }]'
60
61            # Prepare new interest tag.
62            if(t != countInterestTags -1):
63              results += '{"name": "' + str(interestTags[t].description + '", "values": ' +
                    array + '}, ')
64            else:
65              results += '{"name": "' + str(interestTags[t].description + '", "values": ' +
                    array + '}]')
66
67          if 'callback' in request.REQUEST:
68            # Send a JSONP response.
69            data = '%s(%s);' % (request.REQUEST['callback'], results)
70            return HttpResponse(data, "text/javascript")
71
72        else:
73          return HttpResponse('NONE')
74
75      except:
76        logging.debug('request record failed')
77        return HttpResponse('FAILED')
78
79  return HttpResponse(json.dumps(results), "application/json")
```

9.6 Appendix VI - Installation

To run ITSI, the following tools are required:

- Python 2.7.6
- Numpy (required for statistical methods)
- Django 1.7.1 (final)
- Xampp (used version 5.6.3)
- Java Standard Edition

Installation steps on Windows:

1. Install Python
2. Install "pip" by running: "python get-pip.py" in a shell (we provide the script in the installation folder)
3. Run a shell with administrator privileges, execute "pip install Django"
4. Run a shell with administrator privileges, execute "pip install django-extensions"
5. Run a shell with administrator privileges, execute "pip install pyyaml"
6. Install VCForPython27.msi (Python compiler for Windows)
7. Install Xampp
8. Place the "WebService" folder in xampp\htdocs
9. Adapt xampp\htdocs\WebService\mod.wsgi file to fit your Windows installation path
10. Replace the Apache configuration file with the one we provide (check the paths correspond to your Windows installation)
11. Copy mod_wsgi.so in xampp\apache\modules
12. Open a browser and test this link: http://127.0.0.1/hubnet/version

If the link from step 12 opens a web pages with the text "0.1a", then the installation is successful. For administration purposes, we advise to use the Python Django management server as it automatically binds the correct .css files. This can be done by opening a shell, navigate to I:\xampp\htdocs\WebService. Then type the following command: "python .\manage.py runserver 0.0.0.0:80". Finally, to access the wizard, open a new web page with this address: http://127.0.0.1/admin. Then login using the username "admin" and password "livefeed".

9.7 Appendix VII - User manual

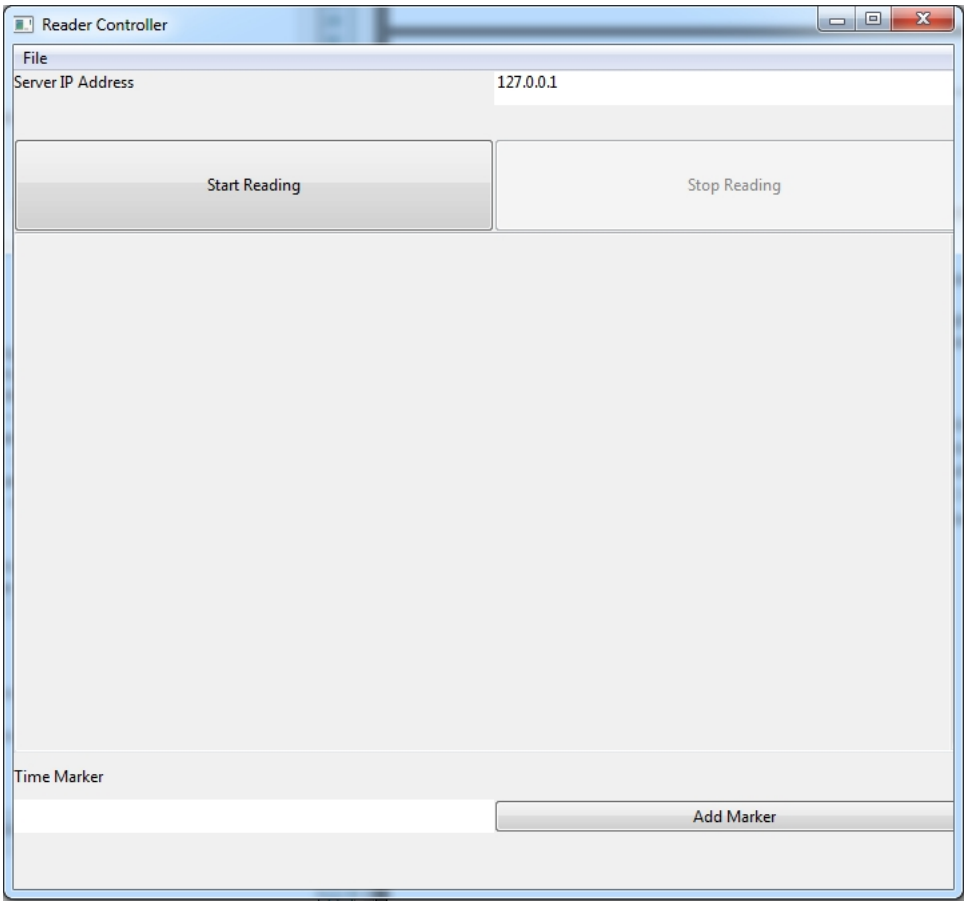
To run a new event, the database has to be prepared. To do so, login to the wizard, and create the following elements:

1. A new event object and fill in the name, start and stop dates.
2. From the same screen, add interest tags, sensors and participants (reuse these or create). You can add more by clicking the "Add another ..." button.
3. Click on the "Save" button.

By doing so, the interest tags, sensors and participants are assigned to the event. The database is now ready to receive the data. The second part consists in configuring the reader controller. To do so, go to the reader controller folder and open the configuration file — default.cfg as an example. This file specifies for each reader its assigned COM port and the id of the event it should record to. Add one line per reader with the following syntax: "com15,3,1," where com15 is the COM port number, 3 the event primary key in the database and 1 the sensor identifier to map the physical reader to. Note that the COM ports can be found in the "Windows Device Manager". It is possible to identify with COM port is assigned to a reader by connecting the readers one by one and identify the newly create COM port in the "Windows Device Manager".

When the configuration file is ready, open the controller.jar program, go to file and load your configuration file by going to file, "Load config". Then, in the GUI fill in the IP address of the web service. From now on, ITSI is configured and can be started using the "Start Reading" button in the controller program, as visible in figure 21.

Figure 21: Controller program GUI



The "Add Marker" button allows to record a new time marker with the text entered in the text box to the left of the button. The middle of the GUI hosts the console. Note that logs can be saved at the

end of an event by going to file, "Save logs". Finally, when the event is over, press the "Stop Reading" button. It will destroy all the reader instances and stand ready to start recording data again using the "Start Reading" button.

Finally, when the controller program is running, data is being recorded to the database. From there the client visualizations can be run. For the live visualization, host the "Visualization" folder provided with an HTTP server. Python provides one by default if required: Open a shell, navigate to the "Visualization" folder and run the following command: "python -m SimpleHTTPServer". Then open a browser and type in the following address: http://127.0.0.1:80/index.html.

Note that the visualization is currently configured to run on the localhost, which is the web service itself. To modify this, change the ip address in the index.html file in all the requests to the web service.