FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Agent-Based Android Application for Conferences Client-Server Communication and Architecture

Thesis supervisor:
Prof. Dr. Ing. Andrei Olaru
Ing. Alexandru Sorici

Alexandru-Dan Tomescu

BUCHAREST

2014

Table of Contents

1 . Introduction....2
1.1. Ambient Intelligence....2
1 .2. Objective: Smart Conferences....4
2 . State of the Art....7
2.1. Conference Management Applications....7
2.2. Multi-Agent Systems....9
3 . The Smart Conference Application....12
3.1. Application Scenarios....12
3.2. Application modules....15
4 . Application Architecture and Implementation....18
4.1. Server Application and Implementation....22
4.2. Client Implementation and Server-Client Communication....25
4.2.1. Server-Client Communication Module....25
4.2.2. Agent Bridge Module....30
5 . Scenario and Testing....32
6 . Conclusion....36
7 . Appendix A....38

ABSTRACT

Ambient Intelligence (AmI) envisions an ubiquitous environment which provides assistance to the people within it. This field has been recently receiving more attention due to the fact that devices are getting smaller and more powerful. The goal of this thesis is to present the context of smart conferences and to implement part of the architecture and specific functionalities of the Envived Android application.

1. Introduction

1.1. Ambient Intelligence

Ambient Intelligence (AmI) is the vision that technology will become invisible, embedded, present whenever we need it, enabled by simple interactions, attuned to our senses and adaptive to users and contexts [1].

The idea of Ambient Intelligence refers to a new way people interact with technology, which is embedded in the environment. Some of the objectives of Ambient Intelligence are the optimization of everyday tasks, the improvement of human communication, comfort, security, health and others. Ambient Intelligence is tied with other technologies including ubiquitous computing and smart environments.

At the core of AmI we have the idea of non-intrusive computing with minimal interaction from the user. This is finally feasible as devices and sensors have become small and powerful enough that they can be concealed within ordinary items though environments, fact which was predicted by Moore's Law. Through these devices and sensors specific environments can be enriched, so that they can react to people and give assistance.

This area of Computer Science also targets to change the way people interact with technology. What started with huge systems with very limited processing power, maintained and used by specialists, transitioned to personal computers which tech-savvy people would operate, then to small devices and wearables (PDAs, Smartphones and recently other wearables such as smart glasses and smart watches), and finally to devices embedded into the environment, which people can interact with no effort.

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." [2]

In the environments enriched with Ami, the user provides input to the devices that surround him with no effort, just by carrying his usual activities. These devices should be able to anticipate the needs of the subject, to interpret his actions and mood, to empathize. This means that the system doesn't just collect data and interacts based on the input, but it is an intelligent system, able to be proactive when needed and to restrain itself in certain situations.

AmI is dependent on other fields in Computer Science like Sensors, Networks, Ubiquitous Computing and Artificial Intelligence.

Sensors are small devices used to detect certain stimuli and measure them. These are used to measure temperature in the environment, the composition of the air and any other measurable quantity in the environment. These sensors provide input for the AmI system, which can then choose a course of action.

Sensors can also be attached to the body of the subject to provide valuable information. One example would be sensors that keep track of the users' pulse and other vital signs. This way the system would be able to provide help by calling an ambulance in case of serious health problems affecting the subject.

Ubiquitous Computing, or Everyware is the computing concept in which computing power is made available everywhere. This concept is based on small processing devices, interconnected through networks.

"Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine starting uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives." Mark Weisser [3]

The tie between Ambient Intelligence and Artificial Intelligence is obvious. Sensors, networks and ubiquitous computing offer the framework for AmI, but the decision making process is provided by the field of Artificial Intelligence. In order for the system to be useful to the subjects it needs to make deductions based on the environment data it gathers and, when needed, to act.

The "5Ws" have been identified and described in the AmI literature [4]: Who, What, Where, When and Why.

Who: this refers to the identification of subjects throughout the environment: people, pets, objects of interest. Another important part of this process is the identification of the relations between the subjects (relations between the people involved and between people and the objects that surround them).

What : involves spatial and temporal awareness and consists of identifying actions performed by subjects in the environment. Based on this the system can provide assistance to the user or can gather data and classify that data in a context. For example if a smart home identifies that a subject is running on a treadmill, it will adapt the heart-rate threshold over which it will call the ambulance. In case the user is just sitting and the heart-rate goes really high, it could mean that the user is having a panic attack and that it is in need of assistance.

Where : involves tracking subjects throughout the environment, which can be done using sensors, and GPS-equipped devices which the user might be wearing (smartphones are very useful here, as most of the time they are kept close to the users, and are usually equipped with GPS trackers).

When : adding the time dimension to the system helps means that it can associate times and durations to activities, giving it a better understanding of what is happening.

Why : understanding the intentions behind activities is really important to an AmI system, as it enables it to be sensible to the needs of the users.

Multiple applications and environments have been defined for AmI: smart homes, smart conferences, smart hospital rooms and others. The objective of these applications is improving the way people function within them.

1.2. Objective: Smart Conferences

For an AmI system to be effective (or even possible at the moment) a context needs to be defined for it, when it is designed. A generic all-purpose AmI system is too complex to build with the present advances in technology.

One context that seems to be very well suited to this vision consists of Smart Conferences. A smart conference consists of a conference setting enriched with an intelligent system which can help the conference attendants with information they need about the schedule or about presentations and other more intelligent tasks like suggesting presentation tracks or meeting people with similar interests.

In the normal conference scenario presentations on certain topics are held in a location containing presentation rooms. The presentations are held by speakers, and sometimes a presentation can have a chair that helps organize and keep order. Depending on the number of presentations and the amount of time allocated, tracks can be set, each containing presentations related to a certain topic or area of discussion.

Attendants register as they arrive and attend the presentations they prefer and during the conference professional relationships are built which can lead to new projects, as the conference attendants probably work in the same field and are interested in similar topics.

The conference scenario (as described above), can be enhanced by adding a digital dimension to the interactions between participants. To aid in this sense, the environment has to be enriched by adding logical (QR codes) and/or physical sensors (iBeacons) in order to detect the location of the attendants. Of course, physical sensors would be optimal, as QR codes depend on whether the user remembers/wants to check-in and it may be considered troublesome by some.

Also, for this scenario to be considered an AmI scenario, it needs to contain a cognitive component: e.g. AI agents located on the participants smartphone devices. Together with the sensors installed through the environment, these agents can facilitate a number of features:

- presentation suggestions based on individual interests
- suggestions about other participants with similar interests and possible topics to discuss
- alerts about schedule modifications
- smartphone automatic settings based on situation (silent mode or automatic message responses during presentations)

- facilitating features for presenters and chairperson (presenter alert in case of being late to his own presentation)
- easy communication between attendants and organizers

Throughout this document, an overview about the state of smart conference applications will be presented, together with smart conference scenarios (which are important as they provide a context for the AmI environment by describing situations and features).

After the general overview of AmI and Smart Conferences, a presentation of the Envived application is made, as the object of this thesis. Envived is a smart conference application consisting of a client and a server side. By combining the advantages of a smartphone with logical or physical sensors and with smart agents, this application brings a new dimension to conferences. An overview of the modules developed as part of this thesis is made.

The objective of this thesis is to provide a clear image on the smart conference scenario, and to contribute to Envived, which puts the concepts into practice. The intention is for the contributions to improve the communication mechanism between server and client and also inter-module (agent bridge). Also, more conference related features would make Envived better suited for conference deployment.

In chapter 2 we present some state of the art contributions in terms of applications and technologies and we discuss how Envived relates to them. Further, in the chapter 3 (The Smart Conference Application), scenarios are modeled for the smart conference application and the main modules are presented. The implementation of these modules is discussed in-depth in chapter 4 (Application Architecture and Implementation). Afterward, chapter 5 (Scenario and Testing) addresses further analysis and testing of each implemented functionality. This thesis concludes with a summary of contributions and a discussion of future work in chapter 6.

2. State of the Art

2.1. Conference Management Applications

As the interest for big conferences has gone up in the last few years, so has the interest in managing them using smart device applications.

In the following, we present a list of conference support applications, detailing their most relevant features and pointing out key elements that are missing.

Groupio[1]

Groupio is available for smart devices (iOS, Android, Blackberry etc) and it focuses on conference management without the artificial intelligence element.

As stated on the applications website, “Groupio makes it convenient for your attendees to access concise and relevant event information when they need it during the event. It also is a part of your event that they take home with them on their device. Your event the way it is with branding, you advertisements, surveys, the capability to message other attendees, and much more.”

The features delivered by this application are:

- *CMS* - the organizer has access to a platform through which he can upload the data containing the participants and other relevant information without going into the technical part of databases.
- *branding* - the opportunity to customize the way the attendees experience the application by changing the app icon, the menu, background and other elements.
- *alerts* - the organizer has the possibility to send popup messages to the attendees to inform them on different aspects. There is also the option to send the messages on fixes dates in the future depending on user timezone.
- *schedule* - the attendee can check the event schedule to see what future presentations are about and to check a description and the bio of the speaker. The events can also be organized as parent-child sessions (much like tracks), and they provide links to resources like pdfs, ppts, websites and surveys.
- *speakers* - each speaker has a profile which the attendees can access to read or to post questions directly. There is also a discussion board style communications way.
- *maps* - integrated with Google Maps and Bing Maps, but also with static maps for interiors.
- *attendee messaging* - attendees can message each other, and each has a profile. Privacy rules can be changed.
- *social* - link to Facebook, Twitter and Google+

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

- *surveys* - the possibility to ask attendees to fill in surveys in order to get feedback and improve the experience in future events.
- *analytics* - charts of how the users used the application and also information on how many times the ads were showed and clicked.

and other.

GenieConnect - SmartConnect[2]

SmartConnect is one of the conference management solutions provided by GenieConnect and it offers the standard management features: designer layout - the possibility to customize the application layout for your conference, smart recommendations - the user can receive recommendations based on the way he uses the application, crowdsourcing - the ability to present surveys and polls to attendees to gather information and feedback, organizer push messaging - the ability to send messages to all the participants' smart devices when needed, gamification - achievements are awarded to some users based on their actions, analytics - graphs and other information about the event, CMS - content management system which can be used by non-technical organizers.

As observed, these conference management applications do not have an Ambient Intelligence side to them, but they only make use of the fact that conference attendees have smart devices on themselves, and serve content through them.

The concept of Smart Conference is much more than that, as it makes use not only of smart devices that the attendees carry with them, but also of physical and logical sensors spread through the environment. The data gathered there is then processed by the system and use to generate new knowledge or to take action when appropriate.

One way to achieve this additional reasoning is through the use of so called Multi-Agent Systems which we focus on next.

2.2. Multi-Agent Systems

Multi-agent systems typically refers to software agents, but there is also the possibility of robots or human teams.

In recent years multi-agent systems have received increasing attention as it turns out that this concept can bring many improvements in certain situations.

Some of the advantages of multi-agent systems are:

- overall system performance improvements

[2]http://www.genie-connect.com/solutions/smartconnect

- resource and computation distribution within a network which means there is no “single point of failure” which makes the system more robust and resistant to failure.
- better way to represent task allocation for different components
- more modular design

In [5] the concept of agent-based software engineering is presented as a way to model complex industrial systems. Complex systems are defined as systems composed of many autonomous parts which interact with each other (each part can be considered an agent within the system).

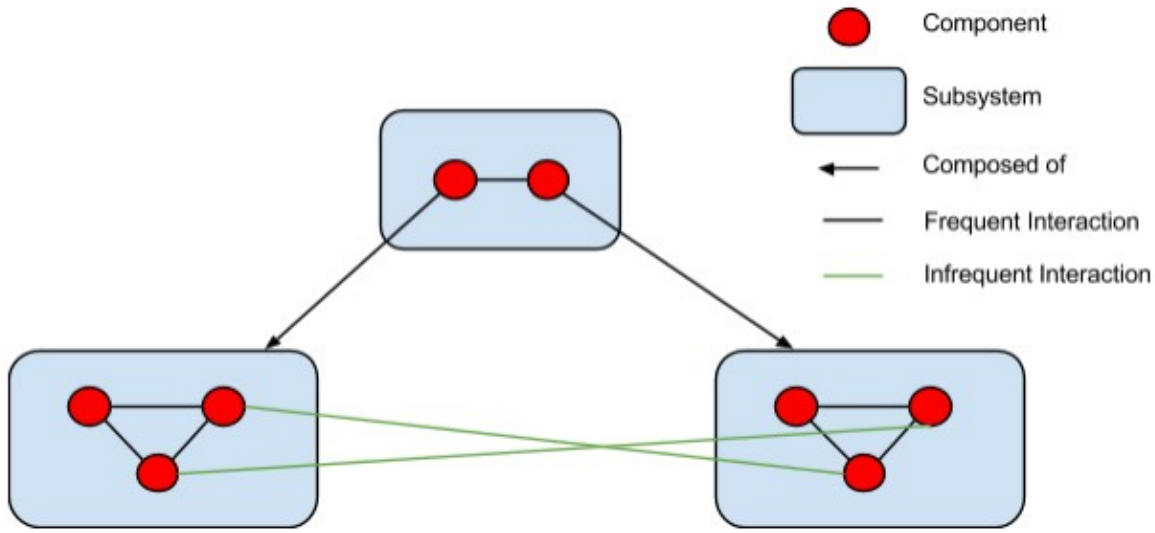


Fig. 1 - The structure of a complex MAS system

There has been a lot of debate on what the best definition of an agent, but seems to be a good candidate: “An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meets its design objectives.” [5] As the definition states, the agent must be part of an environment (virtual, discrete or continuous) and it must have clear objectives which he tries to achieve, cooperating with other agents when needed. Also, the agent has to be autonomous by working on its own and having control over it’s own behavior, it must be able to adopt new goals when needed to achieve the design goal (proactive) and it must be able to respond to changes in the environment (reactive).

One possible context and application for agents is the smart conference context. They enable the intelligent aspect of this scenario by analyzing and disseminating data gathered by sensors - physical or virtual (user preferences) which expand the environment by adding a digital dimension. For instance, when

the speaker of one of the presentations isn't checked in the presentation room at starting time, the agent should send a push notification, or the phone being turned to silent mode when a presentation starts (these scenarios and more will be thoroughly elaborated in the Application Scenario sub-chapter).

The problem of how the agents will behave, coordinate and communicate with each other and the environment is fairly complex so agent architectures have been designed. To give a better understanding of the agent architectures and the differences between them, and to better explain our choice of architecture all the major ones are described below [6]:

Reactive architectures - an architecture which does not need a symbolic model of the environment, and only uses a mapping between situation and actions which the agent can take. Although this method is faster than others that build models, data gathered by sensors might not be enough sometimes. Also, complex tasks might be hard to accomplish as they require lots of mapping of actions to situations.

Logic architectures - architecture which uses a symbolic model of the environment. Decisions are made based on reasoning which leads to good results. The disadvantage of this architecture is that creating or even using the models is a complex task so the system might have poor performance.

Belief, desire, intention architectures (BDI) - architecture which uses philosophy concepts and has known widespread success. On example of this type of architecture is Procedural Reasoning System (PRS) which beside beliefs, desires and intentions also uses two more concepts: plans and interpreter. Beliefs are knowledge which the agent considers true; desires are goal which it has to achieve; intentions are desires which the agent wants to achieve next; plans are the order of actions towards a fulfilling a goal. The interpreter takes all four concepts into account and selects an action.

Layered architectures - a mix between reactive architectures and symbolic model architectures, and it's organized into layers, either vertically or horizontally.

In the smart conference scenario all agent actions are triggered as a reaction to environment changes so the reactive architecture is the most suitable for the agent. Based on this, the tATAmI (towards Agent Technologies for Ambient Intelligence) or Ao Dai platform was chosen. This platform was implemented using a modular structure, and featuring tools for the visualization and tracking of agents, as well as for realization of repeatable experiments, based on repeated scenarios. The platform is underpinned by JADE for the management and mobility of agents and it uses S-CLAIM as language. [7]

In the next chapter, the smart conference scenarios is presented with emphasis on the "smart" aspects facilitated by the agents. Also each module of the project will be detailed to give a perspective on how the application makes the scenarios possible.

3. The Smart Conference Application

A smart conference is one of the most practical implementations of Ambient Intelligence and the topic of this document. In this chapter we want to provide our vision of such an application. We start by presenting a series of interaction episodes which define our envisioned conference application scenario. We then present the high-level architecture overview, describing the main modules of the application and how they are associated with each other. The means by which we added intelligence (the agent component) to the existing Envived conference support application will thus become apparent.

3.1. Application Scenarios

In the case of the smart conference, the environment is enriched with sensors (e.g. physical - iBeacons or logical - QR Codes) and devices (smartphones) which communicate with a system, enriching the conference experience for the attendants.

A short episodic description of the scenarios enabled by Envived is made below:

Preaparation

The attendant (Bob) received an email a few days before the conference asking him to fill up a form containing fields like: interests, affiliation and other. Also, if he is chosen as a session chair, he receives a password which he can use to log in as the chair during the conference. Bob agrees to disclose information about himself, as it is specified that it will be used only in the context of the conference. In the email he also gets a link to a smartphone (iOS, Android or Windows Mobile) application that he can download.

Signup

When Bob arrives at the conference he registers at the reception desk and he tries the application he received. Based on a check-in from a QR code located at the registration booth or based on low-frequency bluetooth beacons, Bob is located within the conference environment.

Schedule

The conference that Bob is attending is one of the biggest AI conferences in the world, which means there are a lot of presentations separated into tracks. To make an informed decision on what he should attend, and not waste time at presentation that do not really interest him, Bob can use the schedule functionality of the smartphone application. Short descriptions about the talk and speaker and tags containing keywords are available. Bob can also choose to mark the presentations that he wants to attend in order to get notifications when one of them starts and he is not localized in the appropriate room.
Based on Bobs’ interests, presentations or tracks can be suggested to him.

Booth visit

The conference that Bob is attending doesn't only offer presentations, but also presentation booths. The interested companies and universities can occupy these booths and offer information on their work or present products of their work. If the booth attendant is busy talking to someone else, Bob doesn't need to wait for his turn, but can use the Envived application to check in at the booth and get access to information about it.

Presentation attendance

When he enters the designated room, he receives a notification that he is attending a presentation, and that the application would like to turn his phone silent, so that it doesn't create disturbances. He can also allow the application to send a preset message to people that are calling him, letting them know that he is busy attending a conference, and that he would get back to them as soon as possible.

Bob also has the possibility to ask questions through the smartphone application: all questions sent this way are presented at the end of the talk, and the speaker can address each of them.

Session chair and speaker functionality

Each presentation has a set time, so that a schedule can be kept. This is held together by the chairperson who announces that a new presentation is to be started, and that the time is soon to run out. If a presenter is not inside the room where his presentation is about to start in just 2-3 minutes, he is automatically notified through the smartphone application that he might be late. When the presenter gets close to the presentation stand where he is appointed, the projects automatically switches to his presentation, and the lights are dimmed for a better viewing experience.

As soon as the presentation is done, the lights are automatically set to their normal intensity, so that people can leave or start asking questions. They also have the possibility to write questions on their smartphones, and they all appear at the end so that the presenter can go through each of them.
If by any reasons the schedule is delayed because of one presentation, it is rearranged automatically or manually by the session chair.

Interest based meetings

Alice has also completed the form with information about her interests, and is attending the conference. Shortly after registering, she is notified through her smartphone application that there is a significant similarity between her profile and Bob's so they should meet. As she was last located at the registration kiosk, and Bob is at a presentation, the application suggests that they should meet in

the break area closest to them. As soon as they both agree to meet, the time is set after the presentation Bob is attending finishes. This way the two attendants get to talk about their projects, and maybe start a joint project for the future.

Organizer messages and notifications

Sometimes the organizers of a conference want to contact specific participants to give them messages (maybe some changes in the schedule for a presentation and the speaker needs to approve it). This can be easily done through the messaging module of the application.

Sometimes coffee and networking breaks are organized in the middle of the conference to give the participants a chance to discuss the topics presented. Everyone can be announce or reminded through a preset notification about the break and the location.

After the conference

Right after the conference Bob has the possibility of viewing the profiles of other participants and maybe get in touch with them about a project that he believes that they may be interested in. He also can save vCards in his phone during the conference so that he can contact them afterwards.

Also, the conference organizers have the possibility to analyze data about how people used the application and which presentations they attended. This way the organizers can extract relevant information and improve the conference for upcoming editions.

Presentation resources

In case Bob is very impressed about one of the talks he can check the application for resources. When a speaker registers a talk he can upload papers and other resources which he believes would be relevant and of interest to the people attending.

The functionalities described above are desired, but in this document I will be addressing a portion of them, the others possibly being addressed in future work. Taking into account the desired functionality described, the main modules of the Envived application are described further below together with the way these address different scenarios.

3.2. Application modules

As Envived is a complex application, dividing it into modules has been an obvious choice. The project consists of two main parts (the client and the server) and each of them is divided into modules:

1. Client
   a) *Android Client Agent* - The agent module which is located on the client

b) *Android Client Agent* - Application Bridge - The module which facilitates communication between the client agent and the client application
c) *Android Client Application Module* - The application client which the user interacts with

2. Server
   a) *Server Agent Module* - The agent module located on the server which synchronizes knowledge between all the client agents
   b) *Server Agent* - Application Bridge - The module which facilitates communication between the server agent and the server application
   c) *Server Application Module* - The RESTful Django application located on the server which, among other functionalities, manages interaction with the database

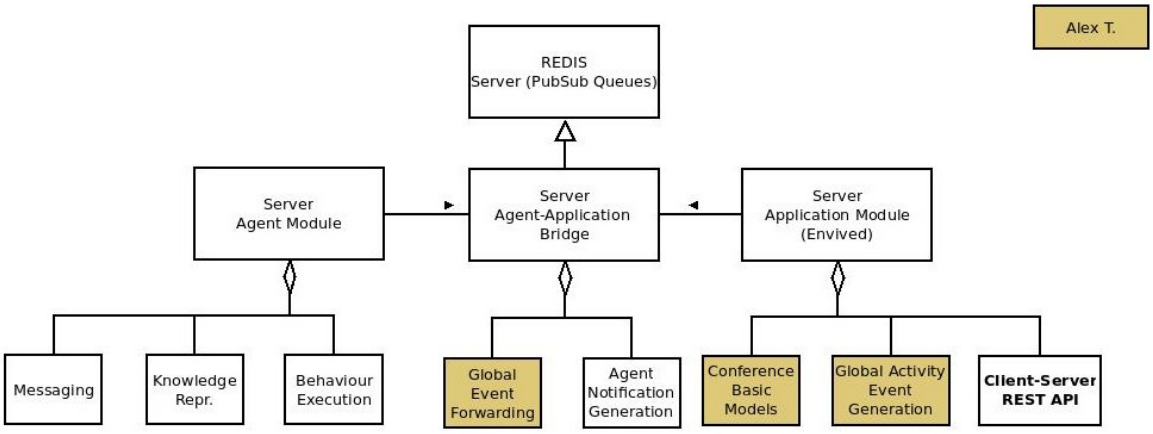


Fig. 2 - Envived Server Architecture - Colored nodes represent contributions of this thesis

As seen in Fig. 2, on the server side, the application consists of three modules: Server Agent Module, Server Agent-Application Bridge Module and Server Application Module. The agent module is the same on both client and server applications and it is implemented through the tATAmI framework[3]. Based on information received from other agents or from the agent bridge module, the agent builds a knowledge base modeled as a labeled graph. The graph is used to facilitate internal agent reasoning based on graph pattern matching. The reasoning process results either in the addition of new edges and nodes to the existing graph (knowledge base update), or in the generation of actions, which are carried out in the environment (the client application acts as the environment or a proxy for the environment).

The agent receives two kinds of messages:

[3]https://github.com/tATAmI-Project

facts - represent information that is considered to be static (does not change during the lifetime of the application) and are always true. This information is usually perceived by the agent module at startup.
events - represent dynamic information which effect changes (additions or removals) in the knowledge graph.

These messages are exchanged between agent and client/server through the agent bridge.

The server application is the module which assures the persistence of data, as the client only presents relevant information to the user and ensures interaction. To this end, data models are used. Also, information is structured into classes using python classes which are used in modeling of the interactions.

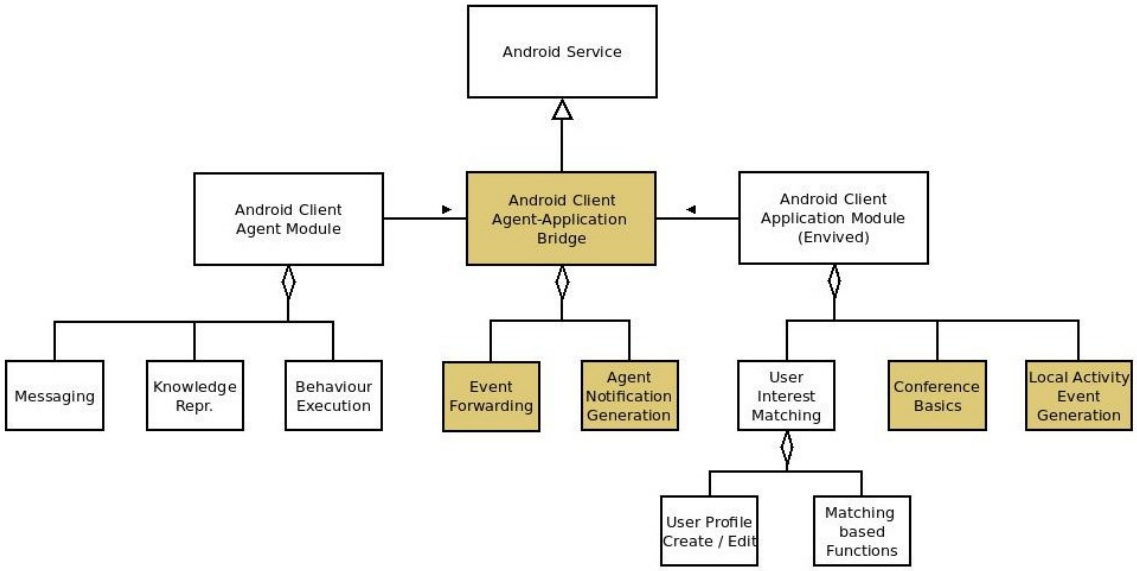


Fig. 3 - Envived Server Architecture - Colored nodes represent contributions of this thesis

On the client side, the agent and agent bridge modules are similar to the ones on the server. The difference is the Android Client Application Module, which contains the user interface and implements communication mechanisms with the server API.

On the server side, there is also an agent implemented through the tATAmI framework. This agent handles the global events and changes, while the client agent manages events that are of concern to that particular client.

4. Application Architecture and Implementation

Envived is designed to be a general application for smart environments, not only smart conference. As a first use and a subject of this document, conference specific features have been developed.

At the base of the design lie a few concepts which make Envived general and modular: environments, areas, features and annotations.

Environments and Areas

Because Envived was conceived with the purpose of creating general smart environments (not only conferences), the first step was to find a way of modeling a physical location and its partitioning. As such, environments model physical locations and, as shown by the parent property in Fig. 4, they can be stand-alone or they can have other environments as parents. Areas model areas of interest within the environments and their definition also contains physical layout description (e.g. shape).

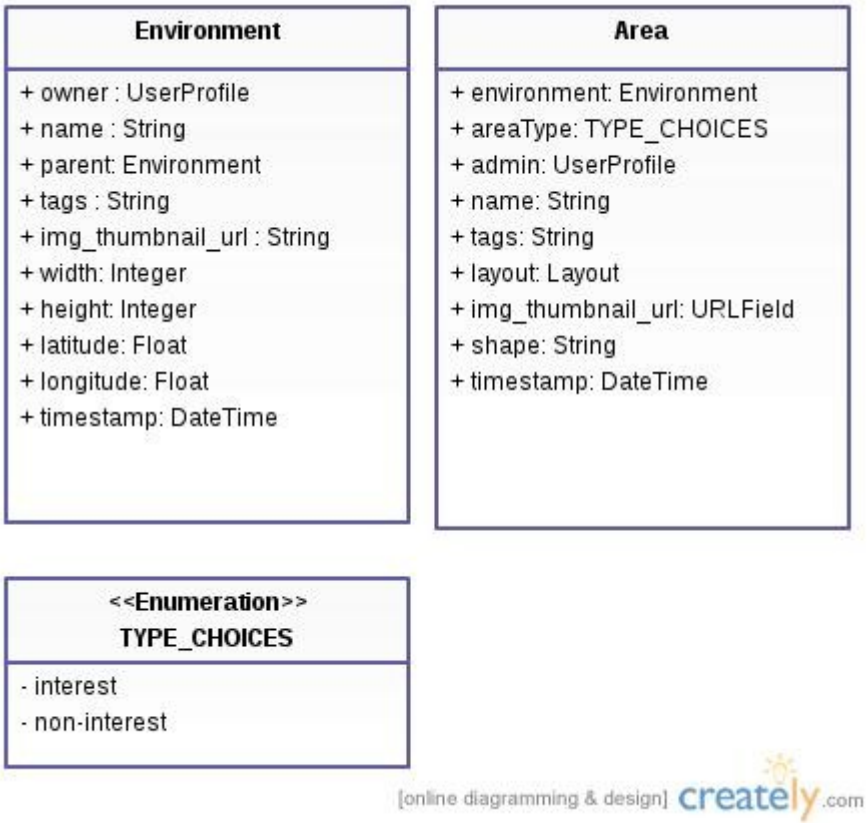


Fig. 4 - Environment and Area Diagram

For example, the AIWO 2013 Conference[4] is an Environment, and each presentation room or booth is an area.

Each area and environment have linked resources on the server-side application and associated features.

On the client-side, Environments and Areas are implemented through the Location class, which has both Environment and Area specific fields and methods.

Features

Features are components which can be attached or detached to an environment. This means that if the application is used for a smart conference, then specific features will be attached, but if it is used for some other purpose, other features will.

Server-side, features manage the way information associated to them and persisted in a database is serialized/deserialized, sent or received from the client, and imprement the corresponding resource views as part of the RESTful client-server communication service (e.g. what to return or what to do when GET, POST, PUT and DELETE requests are received from the client).

Client-side, a Feature base class is defined which contains fields and methods common to all the features:

```
protected Calendar timestamp;   // timestamp of the initialization of the
                                // feature
protected String featureResourceUrl; // URI or the feature resource used by
                                 // requests
protected String environmentUrl;       // URI of the environment to which
                                  // the feature is attached
protected String areaUrl;     // URI of the area within the
                              //environment to which the feature is attached
```

This base class also contains methods for initializing the feature and making GET and POST requests to the corresponding resources.

[4]http://www.aiolympics.ro/

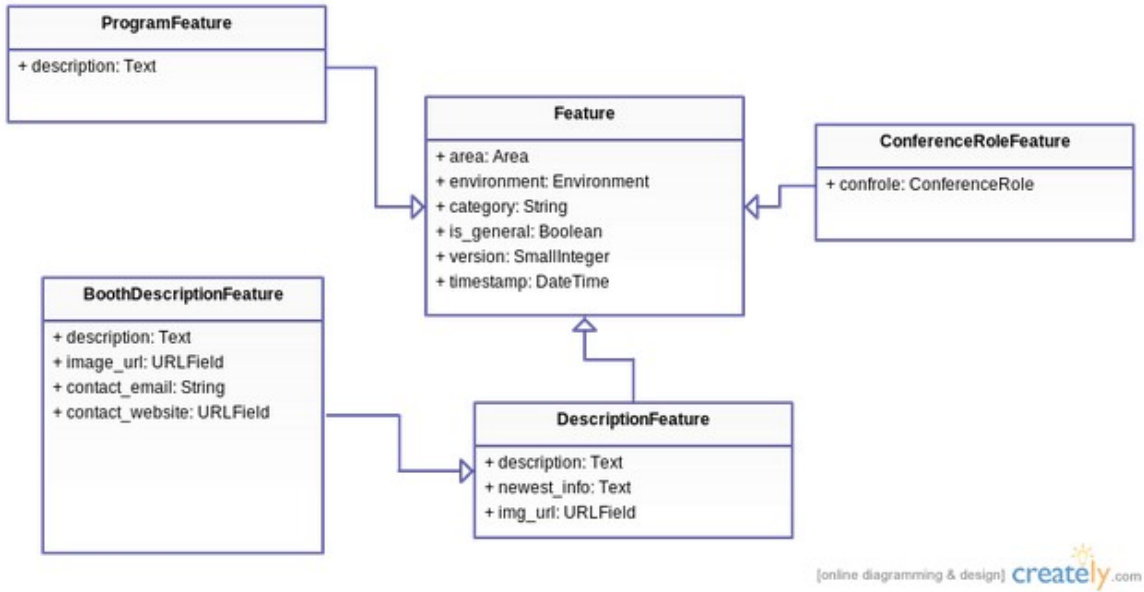


Fig. 5 - Feature Diagram

The way custom features expand the base feature class is illustrated in Fig. 5. The features available for the conference scenario are:

- *Booth Description* - a description page for each booth containing information about the respective booth
- *Program (Conference Schedule)* - all presentations are arranged into a list ordered by time or by session. Also each presentation has a description page containing relevant information (e.g. keywords, description, time of start and end, speaker)
- *Conference Role* - each attendee can choose a role as part of the conference (participant, speaker or session chair)

The conference participant can have three roles:

1. *participant* - simply attends presentations without having any extra duties.
2. *speaker* - can attend presentations but also has to hold one or more of his own
3. *session chair* - hosts presentations, one of the duties being that the chair has to change the time of a presentation in case it gets delayed. The session chair also receives a password that he needs to use in order to have access to the role actions.

When checking into an Environment, the attendee can access the Conference Role feature in order to choose the appropriate role. Other functionality depends (e.g. session chair functionality) in the future on the role chosen by the attendee.

When the user chooses a role, a PUT request is made from the client to the server. The URI used for this request is the features' resource URI. In case the user is not checked in to the environment, or in case the session chair password is wrong, a 401 - Unauthorized error is sent back. Also, in case the JSON sent as

payload through the PUT request is wrong (does not contain the role requested), then a 400 - Bad Request error is sent back. If the request is okay, and the role is changed, a string containing the role of the user is saved in the cached preferences of the application. This way the client doesn't have to make a GET request every time it needs the role.

Annotations

Annotations are one of the core concepts within Envived, and are used in relation to Environments and Areas. Annotations is a base class which contains a reference the environment or area that it is tied to, a reference to the user that generated it, a category and a timestamp. This base class is then extended by other feature-specific classes which contain other information and used as portrayed in Fig. 6 as a way for the client to communicate to the server.

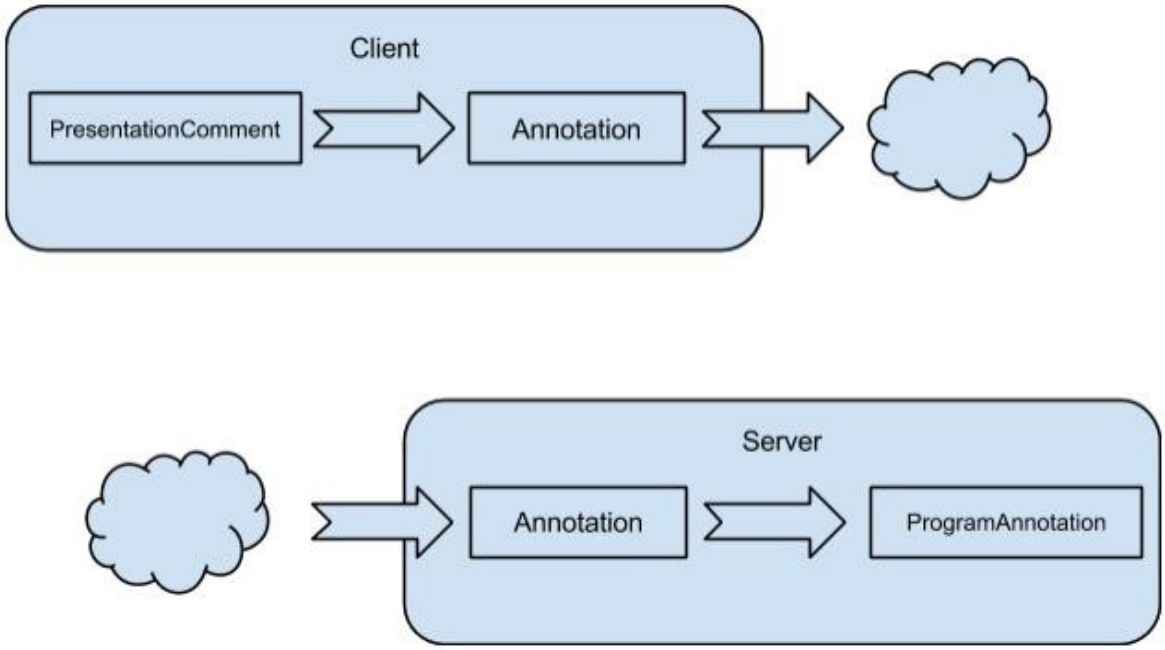


Fig. 6 - Annotation mechanism (from client to server)

One example of the way annotations are used is the comments section in the booth and presentation descriptions. Whenever a user posts a comment in the presentation comments activity, an annotation is generated containing the "booth_description_ann" or "program_ann" category, the location which the comment is in relation to, the timestamp and the comment content, and then it is sent through a POST request to the server.

The server processes the received annotation based on its category and then stores it as a ProgramAnnotation or BoothDescriptionAnnotation, which as shown in Fig. 7, extend the base Annotation class.

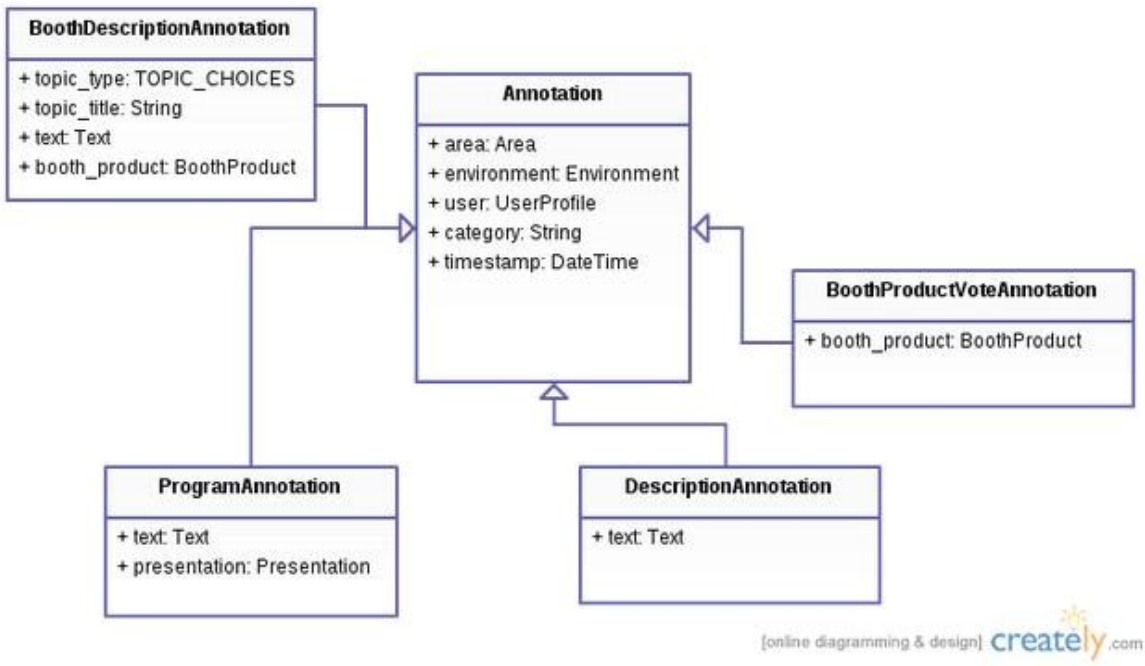


Fig. 7 - Annotation Diagram

Whenever the user starts the comments activity, a request is made for all the annotations related to that location with a certain category. After all the annotations are received, they are parsed and displayed as a list.

4.1. Server Application and Implementation

The server side of the Envived application is implemented in the python language, using the Django framework.

Django is an open source framework which follows the MVC (model-view-controller) pattern. It offers some general modules and functionalities (login/logout, registration or admin panel) and also an efficient way of organizing custom-created modules and functionalities into apps. The apps defined for Envived are:

- agent - the agent logic
- client - implementation of the RESTful API mechanism
- coresql - contains data models and abstract classes used by the features
- features - contains the implementation of the features and extensions of the abstract classes defined in the coresql package
- messaging - implements the agent inboxes and communication which makes the long polling mechanism possible.

From the MVC architectural pattern the View component is less used compared to Model and Controller, as the only front-end component consists of the admin

functionality which is mainly used to add alter and remove records from the database (ex. Environments and Areas).

The Model component is used to create and manipulate the applications' database. The central models.py file (apps/coresql) contains classes which model relational database tables like UserProfile, UserSubProfile, Environment, Area, Announcement, Annotation, History, Privacy and Feature.

Other than the MVC architecture provided by the Django framework, the server application also uses the RESTful architecture in relation to the client.

REST (Representation State Transfer) applications are based on resources and not actions (like in SOAP) and follow a set of constraints[8]:

- Uniform Interface - the interface between the client and the server. In the case of the project discussed in this document, HTTP is used: HTTP verbs are used (GET, POST, UPDATE, DELETE), resources are identified through URIs and the responses are HTTP responses.
- Stateless - the server doesn't know or care about the state of the client, and it's the clients' job to request resources based on it's state and needs,
- Client-Server
- Cachable - all the requested information can be cached on the client either implicitly, explicitly, or based on a negotiation schema
- Layered System

In Envived the REST architectural style is enabled using Tastypie. "Tastypie is a webservice API framework for Django. It provides a convenient, yet powerful and highly customizable abstraction for creating REST-style interfaces." [9]

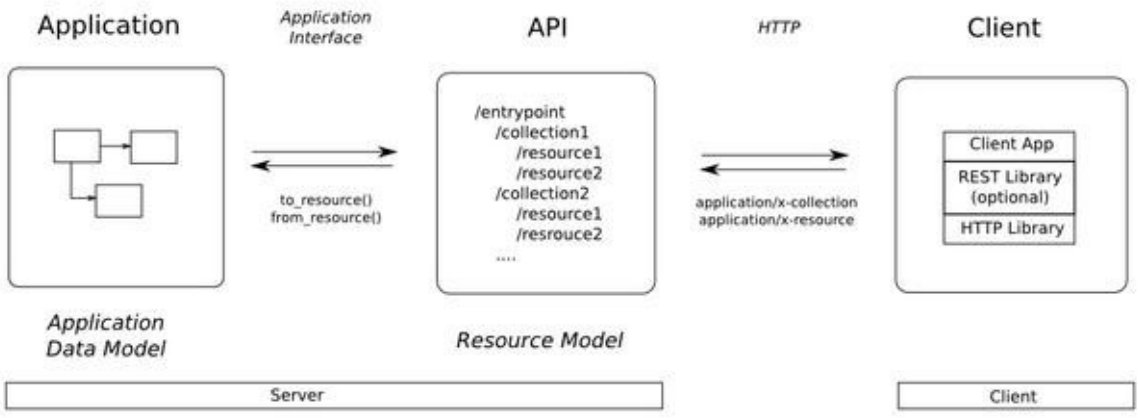


Fig. 8 - Restful API Architecture

As portrayed in Fig. 8[5], the way the server communicates with the client is through the HTTP protocol, based on resources. Each feature is considered a resource, and it corresponds to an URI, so that when the client needs to retrieve

[5] http://restful-api-design.readthedocs.org/en/latest/scope.html

some information from the server it makes a GET request on the resource, and when it needs to send information to the server it uses a PUT HTTP request.

For example, the URI accessed for updating a presentation within a conference program resource could be:
http://localhost:8080/envived/client/v2/resources/features/program/60/
And the payload can contain a JSON with parameters:
{
“old_starting_time”: “12-05-2014 14:00:00”,
“new_starting_time”: “12-05-2014 14:30:00”,
“old_ending_time: “12-05-2014 15:00:00”,
“new_ending_time: “12-05-2014 15:30:00”,
“session_id”: “id”
}, the time parameters being used to identify and change the details of the presentation, and the session id being used to determine which of the presentations should have their time changed.

The resources are defined in the apps/client/api.py file in the server project. Each class models a resources specifying the URLs used and also some metadata to define the way that resource can be used:

```
class Meta:
      allowed_methods = ['get'] # one can only retrieve data from this source
         excludes = ['id', 'is_general'] # the following fields don’t end up in the serialization
      filtering = { # choose the type of filters to be applied on the data
         'area' : ['exact'],
         'environment' : ['exact'],
         'category' : ['exact']
      }
```

The data sent from the server to the client when requested by the client is encoded as JSON. For example when logging in, the user makes a POST request at the following URI containing its username and password:
http://localhost:8080/envived/client/v2/actions/login/

The response from the server is the following JSON:

```
{
  "code":200,
  "data":{
    "first_name":"Alexandru",
    "last_name":"Tomescu",
    "resource_uri":"/envived/client/v2/resources/user/2/"
  },
  "success":true
}
```

4.2. Client Implementation and Server-Client Communication

4.2.1. Server-Client Communication Module

The communication between Server and Client (Server -> Client) is an important module of the whole project, as it is often the case that not only the client requests data, but that the server has to send updates (e.g. when the conference schedule is changed).

While in the case of Client->Server communication, the client can make requests on the resources the server holds, in the Server->Client communication, this cannot be done the same way. The chosen method for this side of the communication process is long polling.

One of the scenarios in which the server needs to send messages to the client is when the conference schedule changes (either changed by one of the session chairs, or changed by someone from the admin interface). When this happens, the server has to send the update (in this case, the new schedule) to all the clients in that environment. There are three types of messages the server can send to the client:

- envived_app_update - messages sent when the content a feature needs to be updated (schedule changes, speakers etc)
- envived_app_message - messages from the server for certain features within the environment
- envived_event - changes in environment that are sent to the client agent, where they are translated into a labeled graph representation form that is consumed by the agent

The Envived client application starts a service called EnvivedMessageService at bootup. This service makes a GET HTTP request to the http://localhost:8080/envived/client/notifications/me/ URL containing a session cookie. Based on the session cookie, the URL is linked to the REDIS queue corresponding to an user.

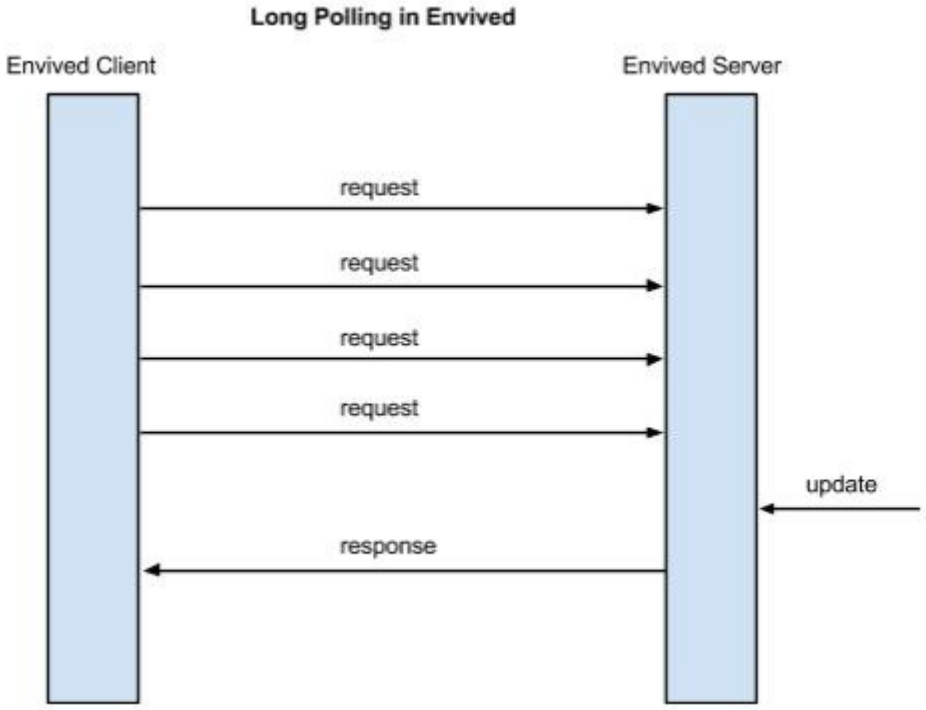


Fig. 9 - Long polling mechanism

After a set amount of time, the GET request times out. If the application is still running, a new GET request is made, as seen in Fig. 9. Whenever the server has a message for the client, a response for the request is received, and the content is forwarded to the class that processes it (e.g. a message sent for the agent is forwarded to the agent bridge). After this, the long polling mechanism is resumed.

The EnvivedMessageService is implemented as an Android Intent Service and the whole long polling logic is contained in the onHandleIntent(Intent arg) method which is called at client startup. This means that when the long polling stops, the service can also stop working.

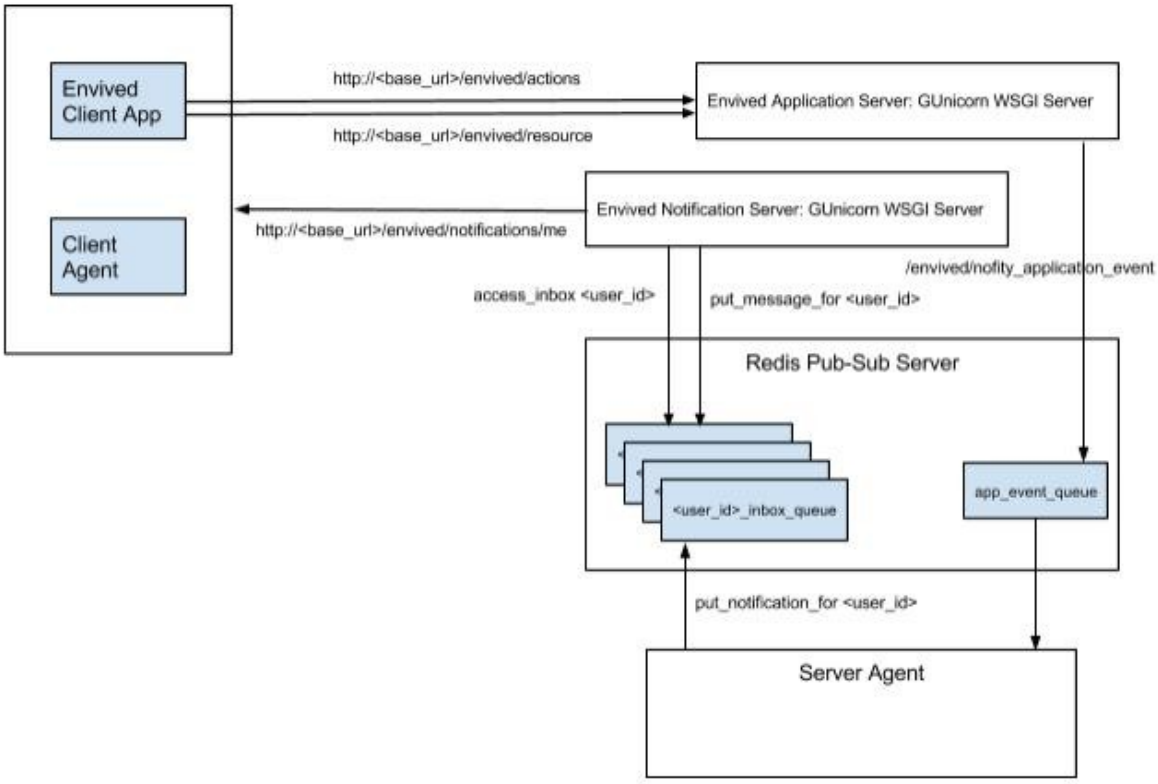


Fig. 10 - Envived Communication

As seen in fig. 10, when the client polls for messages on the http://<base_url>/envived/notifications/me URI, the message is retrieved from a Redis inbox mechanism. This system implements queues for each user which act as inboxes, so that all messages addressed to a specific user are placed there, and retrieved polled for by the client application. Separate from the user inbox queues, an app_event_queue is implemented for the agent so that the server agent can access the event messages it receives. Like in the case when the server application sends updates to clients, when the server agent needs to communicate to the client agents it places the envived_event messages in the user inboxes.

Messages received by the EnvivedMessageService are serialized as JSON:

- envived_app_update and envived_app_message (only the type field differs)

```
{
  "content":{
    "location_uri":"location_uri_example",
    "resource_uri":"resource_uri_example",
    "feature":"program",
    "params":[
      {
        "name":"type",
```

```
        "value":"new_request"
      }
    ]
  },
  "type":"envived_app_update",
  "timestamp":"4-06-2014 15:34"
}
```


- envived_event


```
{
   “facts”: [
      {
         “fact_label”: “fact_label”,
         “subject_label”: “fact_subject_label”,
         “object_label”: “fact_object_label”
      }
   ],
   “events”: [
      {
         “performative”: “INSERT/DELETE”,
         “event_label”: “event_label”,
         “subject_label”: “event_subject_label”,
         “object_label”: “event_object_label”,
      }
   ]
}
```


The envived_app_update message contains updates made within feature data on the server.

This is automatically sent when feature model instances are updated. The Django framework offers the ability to set up function hooks to post database save/update events which allow us to implement the above mentioned automated update notifications.

As an example, the session chair has the possibility to update the schedule of a certain presentation (in case it should start early, or more often, in case it ends later than planned). Whenever a presentation finishes later than planned, the subsequent presentations should also be updated, as they cannot start until the current one finishes, and the changes have to be made on all the clients within the environment.

When the session chair adds time to a presentation, a PUT request is made to the program resource on the server containing a JSON with the old and new start and end times. This way the server can update the model for the respective presentation and to all the subsequent presentations. When the model is saved, a function is triggered which sends the envived_app_update message to all the clients checked in that environment. This message is polled by the Envived Message Service and dispatched to the corresponding feature which updates the local database.

The JSON deserialization is made using the Gson[6] library from Google. This provided a much cleaner and simpler code. Instead of using the java JSONObject and JSONArray classes, this automatically converts the JSON objects into instantiated class objects. The only requirement was to create the three classes: EnvivedAppUpdate, EnvivedAppMessage and EnvivedEvent. The JSON is deserialized based on the names of the fields of the containing classes, and also based on the @SerializedName Java Annotation. If the JSON contains an object or list of objects, such as “facts” or “events” from the envived_event message, containing classes have to be provided (such as com.envived.android.api.agent.Fact and com.envived.android.api.agent.Event).

```
public class Fact implements Serializable {
        private static final long serialVersionUID = 1L;
        @SerializedName("fact_label") private String factLabel;
        @SerializedName("subject_label") private String subjectLabel;
        @SerializedName("object_label") private String objectLabel;
...
}
```

After deserializing the JSON payload from the messages, the new objects are passed on to the activities or services that process them.

In the case of EnvivedAppUpdate and EnvivedAppMessage, these are broadcasted to all dispatcher classes, where they are forwarded to the features for which they are meant for. For instance, EnvivedAppUpdates are sent to EnvivedAppUpdateDispatcher, where features register handlers for certain messages.

EnvivedEvents are not broadcasted, because there is only one class that handles them, and that is EnvivedAgentBridge, so they are sent as an intent to this service.

In order for the EnvivedMessageService service to stop, a flag is set true when the onDestroy() callback method is called. This flag is used as a condition for the long polling loop, so that when the flag is true, it stops and finishes the current onHandleIntent() method.

4.2.2. Agent Bridge Module

The Agent Bridge Module consists of a class containing an Android Service which intermediates communication between the Envived client application and the agent environment. As with the EnvivedMessageService, it extends the Android IntentService class.

When EnvivedMessageService receives a message with the “envived_event” type, then it deserializes it and it passes it to the AgentBridge inside an intent. The EnvivedEvent class contains an ArrayList of Event object and an ArrayList of

[6]https://code.google.com/p/google-gson/

Fact objects. These classes both contain three String labels, and in addition, an Event also contains a performative (action).

Both Event and Fact can be modeled as simple oriented graphs containing two nodes (the object and the subject) connected through an edge with the fact_label or the event_label strings as label.

One possible example is the fact that a user is the speaker for one of the presentations. In this case, the agent would receive the following JSON:

{
 “subject”: <presentation_id>,
 “edge”: “speaker”,
 “object”: <user_id>
}

This translates into the following graph:

<presentation_id> --speaker→ <user_id>

Facts are considered the known information about the environment in which the agent is working, such as the conference program or information about the areas within the environment. Most facts are sent to the agent at startup, in order for it to build its knowledge graph.

Events are new information that comes from the environment and needs to change the agent's current knowledge graph. One example of event that can occur is when a session chair changes the starting or ending time of a presentation, and this change needs to propagate to all the agents in the environment + the server agent. This means that an update is to be performed on the knowledge graph, which is done by deleting the current edges that we want to change, and inserting the updated ones (based on the performative field in the event).

In this chapter, the implementation of the modules and mechanisms of Envived has been presented in order to give more technical insight into the project. Further, the mechanisms described above are illustrated as part of the scenarios added in the third chapter of the thesis (Application Scenarios).

5. Scenario and Testing

After the technical and in-depth exposition of the Envived modules from the fourth chapter of this thesis, the results of this project are presented in relation to the scenarios presented in Section 3.2. Each implemented episode of the scenario is revisited and the interaction with the client application is portrayed through screenshots:

Signup

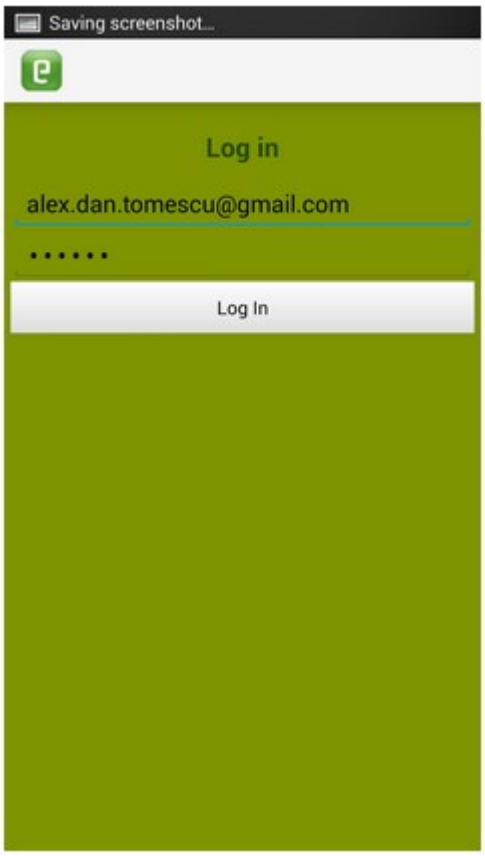


Fig. 11 - Registration Activity

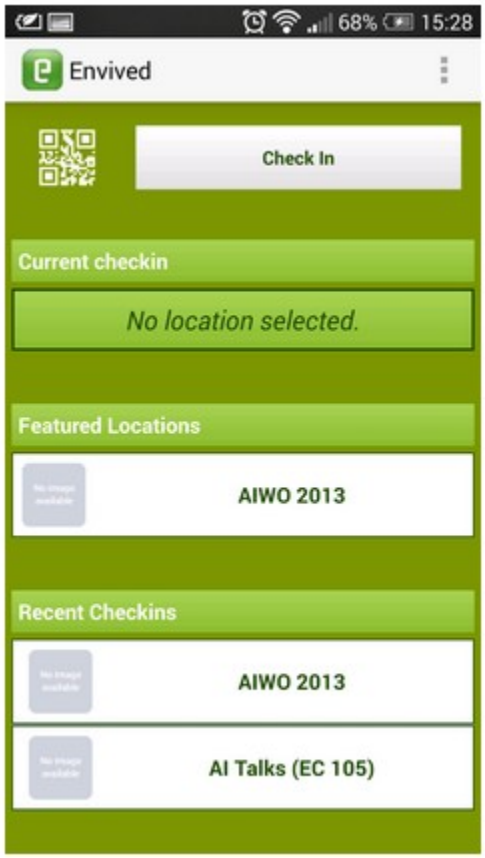


Fig. 12 - Main Dashboard

When the user arrives at the conference where Envived is deployed, he logs in with the account he has created for himself or he has received from the organizers. This takes the user to the main dashboard of the application. As all features are dependent on checking in an environment or location, this step is very important to the overall flow of the application.

The user has two possibilities: a physical checkin or a virtual checkin. In order to physically checkin to a location, the user has to use the “Check In” button which launches the Barcode Scanner[7] application. After scanning the QR code of the targeted location, the application flow directs the user to another activity which

[7]https://play.google.com/store/apps/details?id=com.google.zxing.client.android&hl=ro

contains the available features and functionalities. If the user comes back to this dashboard activity, he can revisit the features list by clicking on the current checkin entry (still considered a physical checkin).

In order to check in virtually, the user can select the conference he is attending from the featured locations section. This means that he has limited access to the features of the conference. Also, the user has the possibility to virtually checkin using the recent checkins section.

Schedule

Fig. 13 - Schedule Activity

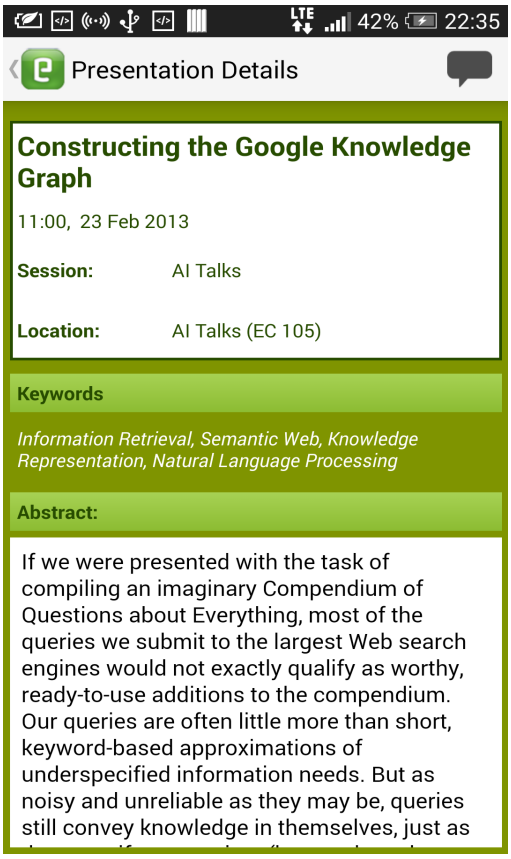


Fig. 14 - Presentation Details

One of the main features that a conference attendee can use is the Program Features. When he wants to check the conference schedule to see the time of one of the presentations that he is interested in, or just to choose a presentation by reading its description, he can navigate to the Program Feature. This feature displays the list of presentations scheduled for this conference, ordered by time or by session. Each entry in the list contains the name of the presentation, the room (Area) where it takes place, the time interval in which it is held, and the session it is a part of.

When the user chooses one of the presentations and clicks on it, a new activity is started containing more information about the respective presentation. Other than the information shown in the previous list, a list of keywords is presented, an abstract for the presentation, as well as a bio of the speaker.

Booth visit

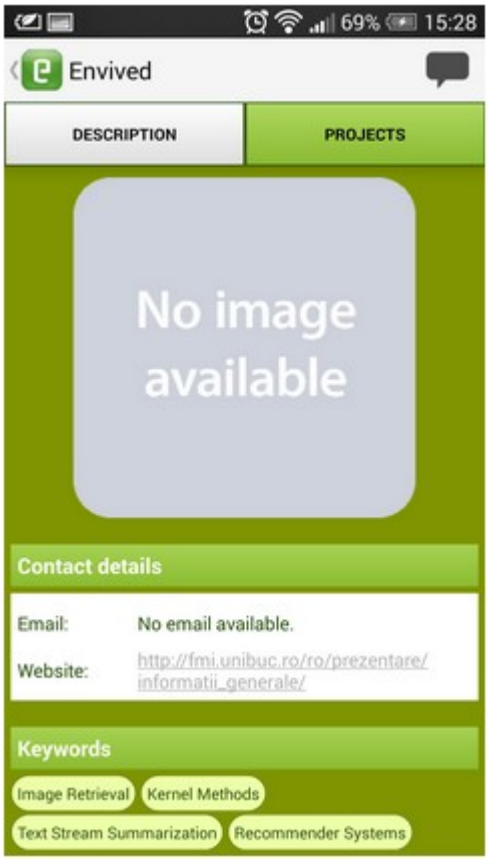


Fig. 15 - Booth Description

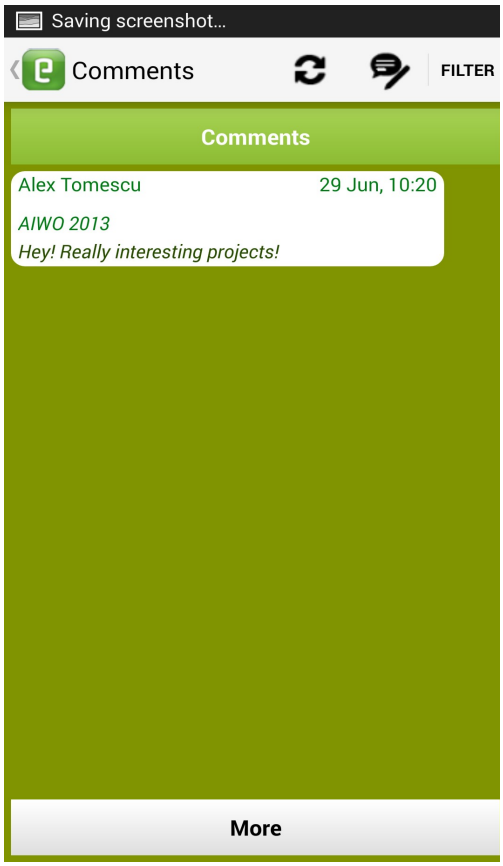


Fig. 16 - Comments Section

In between presentations, the conference attendee can go check the organization booths (if provided). By checking in at the booths in question, the user gains access to their available information (contact details, a description and keywords). A list of projects from the respective company or university is also available. The user can also post a comment to this booth, in order to communicate with other participants or to leave a message for the booth attendant.

Session chair functionality

Fig. 17 - Presentation Details with Session Chair functionality

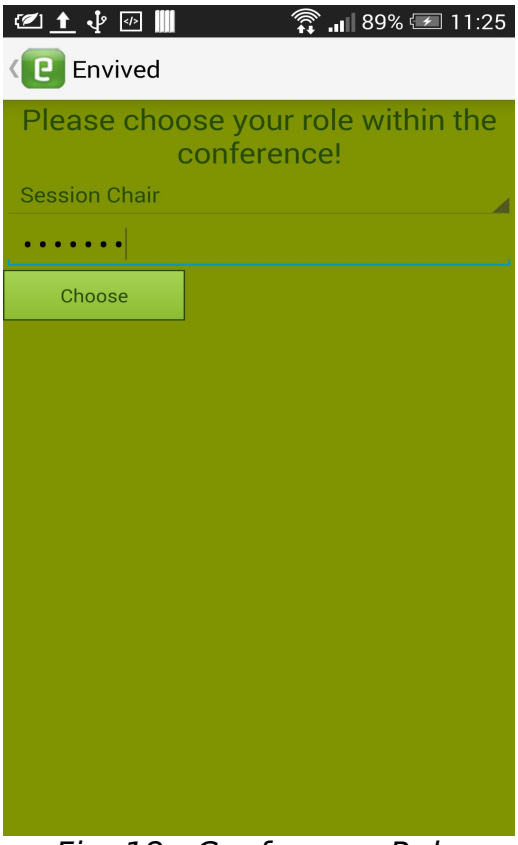


Fig. 18 - Conference Role

If the conference attendee is also a session chair, he receives a password which he can use to log into the respective conference role (the “participant” and “speaker” roles don’t require passwords, as they currently don’t have any special actions associated). After identifying as the session chair, a user can change the starting and ending time of a presentation (most frequently, the ending time, in case a presentation takes more than expected). Whenever the ending time of a presentation is delayed, all the future presentations from that session are delayed, in order not to overlap.

The functionality presented above as part of scenarios is only part of the functionality Envived is meant to deliver. Features like interest-based similarities and suggestions, push notifications from the organizers and bluetooth beacon integration will be tackled in future work.

6. Conclusion

The Envived application is designed to bring a new dimension to the conference context. While there is still work to do, and there are many improvements to be made, Envived can already be deployed in a real conference, making it a more enjoyable experience for the users.

Contribution Overview

The goals related to the Envived project described in the introduction of this thesis have been achieved: the communication between client and server is complete now through the long polling mechanism; the communication between the agent module and the client module is implemented; more conference related functionality has been added, to enrich the users experience. Implementing these components meant working on both the client and the server sides of the application:

On the client side the Agent Bridge Module is the proxy between the Client Agent and the Android Application. When the Application receives a “envived_event” message from the server it hands it over to the Agent Bridge Service which runs in a separate thread in the background. It is then processed and transformed into a graph structure which the Agent can recognize, from which point it can be handled by it.

The mechanism through which the Android Client Application gets messages from the Server Application is long polling: the client starts a separate thread service which continuously polls for messages on the inbox URI of the logged in user. The request made by the client has a well-defined timeout, after which, if the service was not stopped (in case the application was stopped), retries. When a message is received from the server, depending on its type, it is either broadcasted as an intent so that all the interested receivers get it, or it is sent to the Agent Bridge (in case of “envived_event” messages).

Some features and functionalities that were implemented meant working on both the agent and the client applications.

For instance, the Conference Role features is one contribution which allows the user to choose a role when joining a conference: participant, speaker or session chair. The participant role is the basic role, and might not presume any special functionalities, but the speaker and session chair roles indicate that a user has certain responsibilities, which Envived can facilitate.

For the case of the session chair, the program feature allows the specific action of changing the time of a presentation. In case the ending time of a presentation is delayed, all other presentations in the same session are also delayed to make sure they don’t overlap.

One other functionality consists of the comments sections in the program description and the booth description. When a user wants to communicate something to the other participants or organizers, concerning one of the booths

or one of the presentations (before, during or after), the user can access the comments activity and post there. On creation of the activity, all the annotations associated with that location and that category are retrieved through a request. These annotations contain the message, the timestamp and the author of the comment and so are processed and displayed. Whenever a user wants to post a message an annotation is created and sent to the server, where it is stored.

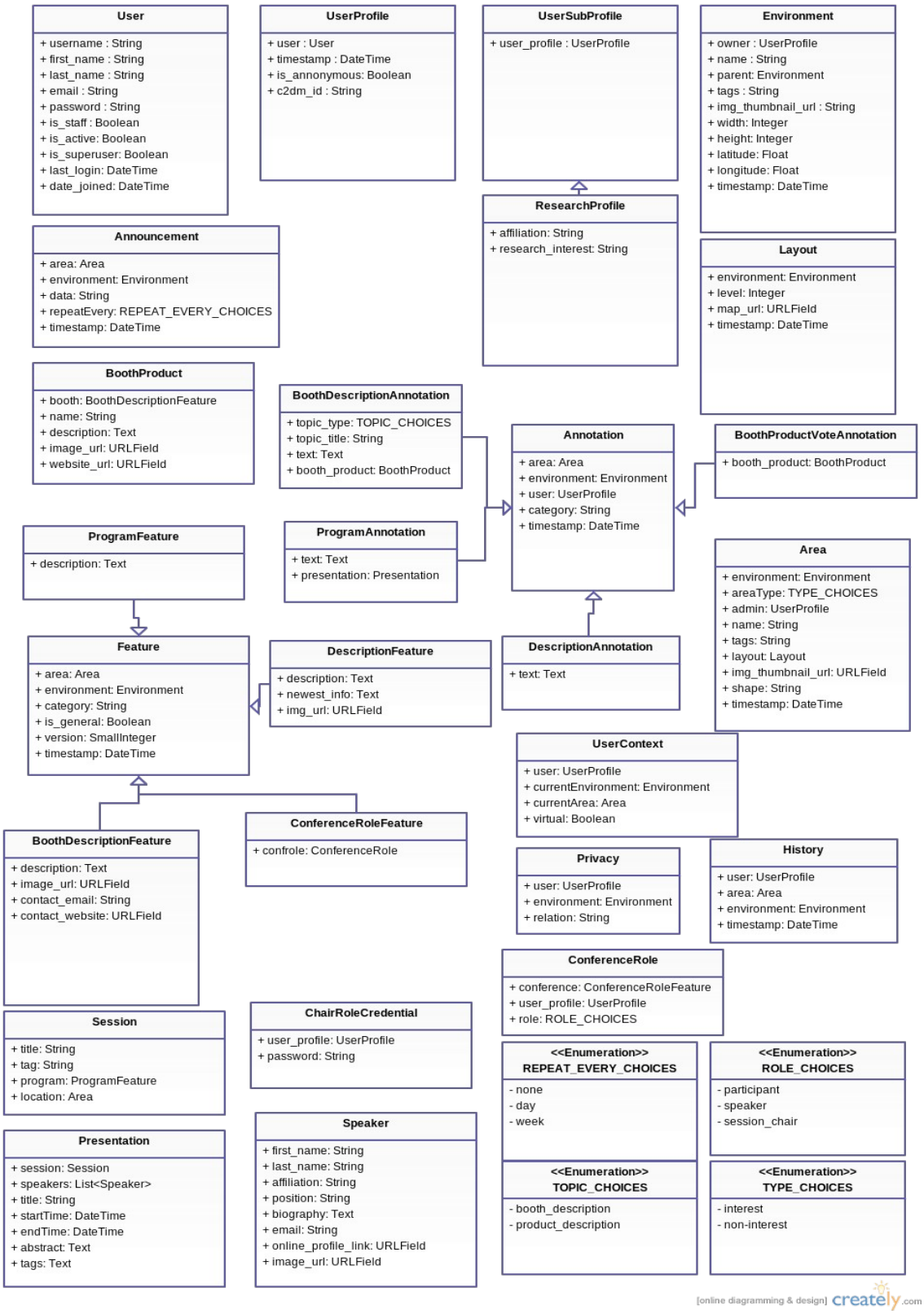
The functionalities above are the ones we have worked on as part of this project. At this point Envived is close to the other smart conference applications on the market, but it can differentiate itself through a few upcoming functionalities.

Future Work

One major improvement would be the use of interest based similarities and suggestions, which would certainly improve the “smart” aspect. Based on information provided by the user and collected by sensors and from the way the application is used, a profile can be made for each attendee and smart suggestions can be made.

Another important improvement can be the use of physical checkins instead of virtual ones through beacon devices: iBeacon or Estimote beacons. These beacons send a low frequency bluetooth signal, so they can be placed into the different areas at the conference, and when the smartphone of a user receives a signal from one of the beacons it can conclude that the user is in that room. This adds context to the application without the user having to go through the hassle of manually checking in every time they enter a different area.

APPENDIX A


User
+ username : String
+ first_name : String
+ last_name : String
+ email : String
+ password : String
+ is_staff : Boolean
+ is_active: Boolean
+ is_superuser: Boolean
+ last_login: DateTime
+ date_joined: DateTime
UserProfile
+ user : User
+ timestamp : DateTime
+ is_annonymous: Boolean
+ c2dm_id : String
UserSubProfile
+ user_profile : UserProfile
ResearchProfile
+ affiliation: String
+ research_interest: String
Environment
+ owner : UserProfile
+ name : String
+ parent: Environment
+ tags : String
+ img_thumbnail_url : String
+ width: Integer
+ height: Integer
+ latitude: Float
+ longitude: Float
+ timestamp: DateTime
Announcement
+ area: Area
+ environment: Environment
+ data: String
+ repeatEvery: REPEAT_EVERY_CHOICES
+ timestamp: DateTime
Layout
+ environment: Environment
+ level: Integer
+ map_url: URLField
+ timestamp: DateTime
BoothProduct
+ booth: BoothDescriptionFeature
+ name: String
+ description: Text
+ image_url: URLField
+ website_url: URLField
BoothDescriptionAnnotation
+ topic_type: TOPIC_CHOICES
+ topic_title: String
+ text: Text
+ booth_product: BoothProduct
Annotation
+ area: Area
+ environment: Environment
+ user: UserProfile
+ category: String
+ timestamp: DateTime
BoothProductVoteAnnotation
+ booth_product: BoothProduct
ProgramFeature
+ description: Text
ProgramAnnotation
+ text: Text
+ presentation: Presentation
Area
+ environment: Environment
+ areaType: TYPE_CHOICES
+ admin: UserProfile
+ name: String
+ tags: String
+ layout: Layout
+ img_thumbnail_url: URLField
+ shape: String
+ timestamp: DateTime
Feature
+ area: Area
+ environment: Environment
+ category: String
+ is_general: Boolean
+ version: SmallInteger
+ timestamp: DateTime
DescriptionFeature
+ description: Text
+ newest_info: Text
+ img_url: URLField
DescriptionAnnotation
+ text: Text
UserContext
+ user: UserProfile
+ currentEnvironment: Environment
+ currentArea: Area
+ virtual: Boolean
BoothDescriptionFeature
+ description: Text
+ image_url: URLField
+ contact_email: String
+ contact_website: URLField
ConferenceRoleFeature
+ confrole: ConferenceRole
Privacy
+ user: UserProfile
+ environment: Environment
+ relation: String
History
+ user: UserProfile
+ area: Area
+ environment: Environment
+ timestamp: DateTime
ConferenceRole
+ conference: ConferenceRoleFeature
+ user_profile: UserProfile
+ role: ROLE_CHOICES
Session
+ title: String
+ tag: String
+ program: ProgramFeature
+ location: Area
ChairRoleCredential
+ user_profile: UserProfile
+ password: String
<<Enumeration>>
REPEAT_EVERY_CHOICES
- none
- day
- week
<<Enumeration>>
ROLE_CHOICES
- participant
- speaker
- session_chair
Speaker
+ first_name: String
+ last_name: String
+ affiliation: String
+ position: String
+ biography: Text
+ email: String
+ online_profile_link: URLField
+ image_url: URLField
Presentation
+ session: Session
+ speakers: List<Speaker>
+ title: String
+ startTime: DateTime
+ endTime: DateTime
+ abstract: Text
+ tags: Text
<<Enumeration>>
TOPIC_CHOICES
- booth_description
- product_description
<<Enumeration>>
TYPE_CHOICES
- interest
- non-interest
[online diagramming & design] creately.com

7. Bibliography

[1] - Ducatel, Ken. Scenarios for Ambient Intelligence in 2010: Final Report; February 2001. Luxembourg: Office for Official Publications of the European Communities, 2001. Print.

[2] - Weiser, Mark. "The Computer for the 21st Century." Scientific American 265.3 (1991): 94-104. Print.

[3] - "Ubiquitous Computing."Ubiquitous Computing. Web. 02 Apr. 2014. <http://www-sul.stanford.edu/weiser/Ubiq.html>

[4] - K. Brooks, “The context quintet: narrative elements applied to context awareness”. Proceedings of the International Conference on Human Computer Interaction. Erlbaum Associates, Inc., 2003.

[5] - Nicholas R. Jennings and Michael Wooldridge, “Agent-Oriented Software Engineering”

[6] - Fabio Bellifemine, Giovanni Caire and Dominic Greenwood, “Developing Multi-Agent Systems with JADE”

[7] - A Context-Aware Multi-Agent System as a Middleware for Ambient Intelligence, Andrei Olaru, Adina Magda Florea, Amal El Fallah Seghrouchni