UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DEVELOPMENT OF A SEMANTIC CONTEXTUAL CONTENT MANAGEMENT SYSTEM FOR MOBILE DEVICES IN BEACON POWERED ENVIRONMENTS APPLICATION IN A SMART OFFICE SCENARIO

Javier Ruiz Corisco

Enero de 2017

TRABAJO FIN DE GRADO

Título:	Desarrollo de un sistema semántico contextual de gestión de contenidos para dispositivos móviles en entornos con Beacons.
Título (inglés):	Development of a Semantic Contextual Content Management System for Mobile Devices in Beacon Powered Environments. Application in a Smart Office Scenario.
Autor:	Javier Ruiz Corisco
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Juan Quemada Vives
Vocal:	Joaquín Salvachúa Rodríguez
Secretario:	Gabriel Huecas Fernández-Toribio
Suplente:	Santiago Pavón Gómez

FECHA DE LECTURA: 26 de Enero de 2017

CALIFICACIÓN:

Resumen

La tecnología está cambiando día a día la manera en la que vivimos. La constante evolución de la tecnología implica un cambio también en la manera en la que la gente trabaja. Las últimas tendencias, como el "Internet de las Cosas", están transformando nuestro entorno y la manera en la que consumimos contenido desde nuestros dispositivos. Los lugares de trabajo no son una excepción, y el concepto de "oficina inteligente" está muy presente hoy en día, permitiéndonos hacer el lugar físico de trabajo más inteligente y adaptable.

El objetivo de este proyecto es el desarollo de un gestor de contenido contextual, y su integración en una plataforma de automatización de tareas. El proyecto proveerá a los usuarios de contenido basándose en eventos con distinto origen y contexto. Estos contenidos pueden redifinir la manera en la que los usuarios interactúan con su lugar de trabajo, teniendo un impacto positivo en su rendimiento y mejorando en general la productividad.

Los usuarios pueden crear contenido personalizado a través del gestor de contenido. Mediante el uso del editor gráfico los usuarios podrán hacerlo de manera fácil e intuitiva.

El sistema ha sido integrado en una plataforma de automatización de tareas para poder entregar el contenido basándose en tecnologías semánticas. Para lograrlo se han definido en esta plataforma nuevos canales, reglas y una ontología.

Mediante el uso de una aplicación Android los eventos originados desde beacons o de otros canales son mandados a la plataforma de automatización de tareas para su evaluación. Los usuarios recibirán una notificación en su dispositivo si existe contenido disponible para ellos, dependiendo del contexto.

El proyecto ha sido aplicado en el entorno de una oficina inteligente, permitiendo su uso en diferentes casos como la mejora de navegación en el lugar de trabajo o alertas sobre eventos de la compañía.

Por último se presentan las conclusiones extraídas de este proyecto, los problemas encontrados durante su desarrollo y las posibles líneas de trabajos futuros.

Palabras clave: Gestor de contenidos, Tecnología semántica, Contexto, Automatización de Tareas, Beacons, Oficina Inteligente

Abstract

Technology is constantly changing the way we live our lives. The constant evolution of technology means the way people do business is also changing. Trending technologies, like the Internet of Things (IoT), are transforming our environment and the way we consume content through our devices. Workplaces are no exception, and the concept of "smart office" is very present nowadays, allowing us to make the physical work environment more intelligent and adaptable.

The objective of this final project thesis is the development of a context aware content management system and its integration into a task automation platform. The project will provide users with custom content based on contextual event sources. These contents can redefine the way users interact with their workplace, having a positive impact on their performance and boosting productivity overall.

Users can create their own custom content through the content management system. Making use of the graphical editor, users can create new content in an easy way.

The system has been integrated into a task automation platform in order to enable context-aware content delivery based on semantic technologies. For this purpose we have defined new channels, rules and an ontology.

A mobile application has been used in order to capture events from beacons or other channels, sending them to the Task Automation platform for evaluation. Users will get notified on their phones if there is content available for them, depending on their context.

The project has been evaluated in a smart office environment, allowing its use for different cases, such as enhanced o ce navigation or company events announcements.

Finally, we present the conclusions drawn from this project, the problems that we have faced during the development and possible future lines of work.

Keywords: Custom Content, Semantics, Context, Automation, Beacons, Smart Office

Agradecimientos

A mis padres, por su apoyo, su confianza y su paciencia. Sin ellos no habría llegado hasta aquí. Gracias por enseñarme y darme tanto, sois el mejor ejemplo de personas al que alguien podría aspirar a llegar algún día.

A los que me han ayudado y han estado desde el momento en el que entré en la Escuela, y por supuesto a los que han aparecido al final cuando más lo necesitaba para darme ese ultimo empujón.

Y en especial a ella, por hacerme grande y darle luz a mis días más oscuros.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

The constant evolution of technology has changed the way we live. Trending technologies, like the Internet of Things, have the potential to impact our routine. The "smart city" concept is becoming a reality all over the world, looking for ways to improve efficiency on things such as energy use. It is natural that the "smart" concept is starting to also being applied to workplaces.

The main goal of the smart office is to make the physical workplace more intelligent and adaptable to the objectives and the operative of the company. By making the workplace interact with the users, and not only the other way around, employees can free more time in order to do real work – the work that makes the employee more valuable. Interacting with user smartphones is the best way to do so.

Customizing the way the workplace interacts with users is a start, but letting users customize it the way they want it is great. It avoids having them feeling out of control. There has been an increase in the need of automating time-consuming everyday tasks. The proof of it is in the number of services that offer this feature such as Zapier [10]. This is accomplished by using Event-Condition-Action (ECA) rules. Providing an intuitive visual interface, users

without programming knowledge can establish their own rules interconnecting different applications.

These systems are called Task Automation Services (TAS), and they are a solid choice when it comes to empower smart offices. By allowing users to create and manage their own automations we will make them happier and more productive.

1.2 Project goals

In this context, the objective of this final project is the development of a content management system to deliver contextual information to mobile devices in beacon powered environments, such as Smart Offices. In order to achieve this goal, the project defines the following subgoals:

- Development of a Content Management System (CMS) that provides authoring tools for assigning contents depending on the user context.
- Development of a Content Delivery Service, in order to integrate the CMS in the platform EWETasker [6] developed by the Intelligent Systems Group (GSI) of UPM. This integration will include: (i) definition of a service of the CMS to integrate with the rule engine; and, (ii) enabling the rule engine to execute content selection rules.
- Integration with the Android client that interacts with the iBeacon enabled space and receives the actions of the rule engine. In particular, a new action will be defined to present users the contents as notifications.
- Evaluation of the project in a smart office scenario.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is the following:
Chapter 1 explains the context in which this project is developed. Furthermore, it describes the main goals to achieve in this project.
Chapter 2 describes the main technologies evaluated in order to design and implement this project.
Chapter 3 goes into detail about the architecture of this project, including the design phase and implementation details.

Chapter 4 provides an evaluation of the project in a smart office scenario.

Chapter 5 gathers the conclusions obtained from this project and provides some suggestions for possible future work to improve it.

CHAPTER 2

Enabling Technologies

In this chapter, we are going to explain the different approaches and technologies that we have used in the making of this project. In the first place, we will start with the tools used for our custom Content Management System, such as SirTrevor for the actual content system and a document-oriented database, MongoDB. After that, we will give a brief overview of the technology developed by Intelligent Systems Group (GSI) where this project will be integrated.

2.1 Beacons

Beacons are BLE (Bluetooth-Low-Energy) powered devices that broadcast small data packets. It is really important to know that the broadcast takes place at regular time intervals and not continuously. This allows beacons to operate on really small batteries, such as coin-cell or AA ones, for several months. Increasing the interval time can help to increase the battery life on the beacons given there is no need for constant transmission all the time.

Smart Bluetooth devices, such as smartphones, use the beacon's signal strength in order to estimate the distance between them. Typically the range will be about 50 meters.

For this project we will be working with Estimote Beacons [6] because of the good results

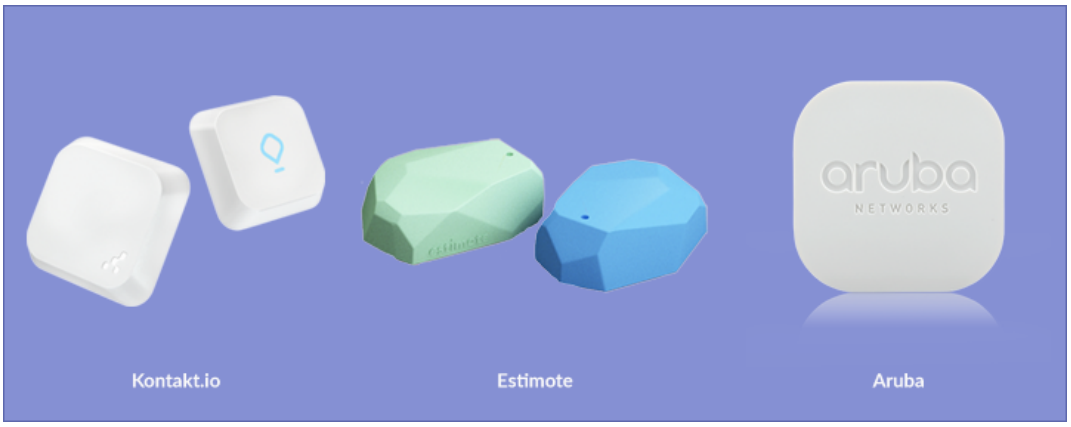

Figure 2.1: Commercial Bluetooth Beacon Solutions

that the Intelligent Systems Group has been getting over the past few years. There is an increasing market emerging around the beacons but Estimote has been a solid choice for us. Some of the reasons why Estimote is a great choice for anyone looking to work on beacons are:

- You can get to 9.83 out of 10 smartphones with it since it's available for both Android and iOS [4].
- Broadcast interval can be customized depending on your needs.
- Great design, making it perfect for any environment.
- Battery life can last up to two years.

Estimote Android SDK

Estimote offers a well-documented library in order to make our apps interact with the beacons. Estimote offers both an Android and an iOS SDK. In our case, and since we are integrating into the Intelligent Systems Group Android Application we will use the Android one. This allows for:

- Beacon monitoring, so you can monitor a region and the devices that enter or exit it.
- Beacon ranging, for debugging purposes to manage all your beacons.

2.2 Task Automation Server

In order to deliver the content to the users in this project, we are going to need some sort of automation. Depending on the context we will show one type of context or another, and to do so the best way is making use of Rules. So we are going to use a Semantic Rule Task Automation Engine called EWE-Tasker. In this section we will go over the engine itself, and the tools that power it.

2.2.1 EWE-Tasker

EWE-Tasker [15] is a web application (created by Sergio Muñoz López, from the Intelligent Systems Group) which does Semantic Rule Task Automation. It uses the EWE Ontology (more on 2.2.4), based on Notation3, to specify these rules.

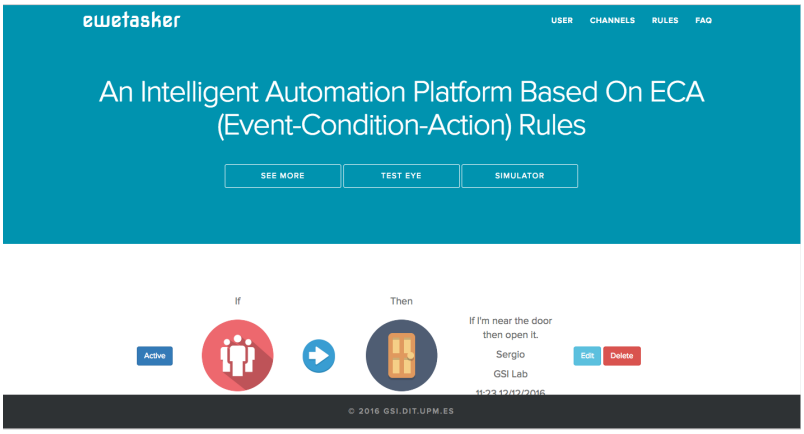

Figure 2.2: EWE-Tasker

These rules have a Event - Condition - Action structure meaning that when an event takes place, a condition is evaluated and if needed an action is triggered. Both events and actions belong to channels that can be easily managed by users. With EWE-Tasker users can:

- Create and import rules for a specific task automation.
- Create and modify the channels that contain events and actions.
- Test rules with the EYE reasoner.

2.2.2 Notation3

Notation3 [18], also known as N3, is a semantic language designed with human-readability in mind. Its compact and readable features make it a good alternative to RDF's standard XML syntax. The language has been developed by Tim Berners-Lee and more members from the Semantic Web community.

The aims of the language are:

- To optimize expression of data and logic in the same language.
- To allow RDF to be expressed.
- To allow rules to be integrated smoothly with RDF.
- To allow quoting so that statements about statements can be made.
- To be as readable, natural, and symmetrical as possible.

The language achieves these objectives with the following features:

- URI abbreviation using prefixes which are bound to a namespace (using @prefix) a bit like in XML.
- Repetition of another object for the same subject and predicate using a comma ",".
- Repetition of another predicate for the same subject using a semicolon ";".
- Bnode syntax with a certain properties just put the properties between [and].
- Formulae allowing N3 graphs to be quoted within N3 graphs using and .
- Variables and quantification to allow rules, etc to be expressed.
- A simple and consistent grammar.

2.2.3 EYE

EYE [11] stands for "Euler Yet another proof Engine", and it is a a high-performance reasoning engine that uses an optimized resolution principle, supporting forward and backward reasoning and Euler path detection to avoid loops in an inference graph. It is written in Prolog and supports, among others, all built-in predicates defined in the Prolog ISO standard. Backward reasoning with new variables in the head of a rule and list predicates are a

useful plus when dealing with OWL ontologies, so is more expressive than RDFox or FuXi, whilst being more performant than other N3 reasoners.

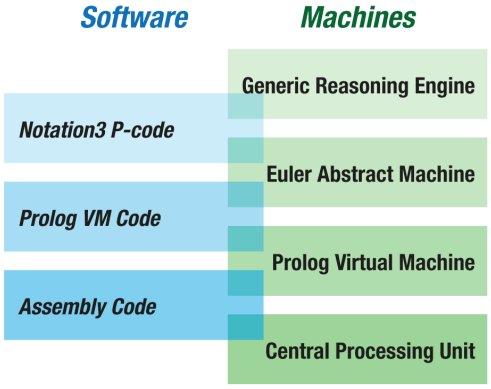

Figure 2.3: EYE Integration

Internally, EYE translates the supported rule language, N3, to Prolog Coherent Logic intermediate code and runs it on YAP (Yet Another Prolog) engine, a high performance Prolog compiler for demand-driven indexing. The inference engine supports monotonic abduction-deduction-induction reasoning cycle. EYE can be configured with many options of reasoning, such as not providing false model, output filtering, and can also provide useful information of reasoning, for example, proof explanation, debugging logs, and warning logs. The inference engine can be added new features by using user-defined plugins.

2.2.4 EWE Ontology

Evented Web Ontology [3], also known as EWE, is an ontology designed to describe elements within Task Automation Services from a descriptive approach, enabling rule interoperability. EWE ontology is composed of four main classes:

- **Event:** The event class defines a particular occurrence of a process generated by a particular service. These events are instantaneous, and let the users describe under which conditions they should be triggered. Changes on the state of a system, or a sensor (like proximity to a beacon) can be modeled as events.

- **Action:** Action class defines an operation or process being provided by a service. They are the result of a rule, this being composed of an event as an input and an action as an output. We can think of the Action class as the output of a Rule.

- **Channel:** A Channel defines an individual which either generates Events, provide Actions, or both. So sensors and actuators are both described as channels.
- **Rule:** The Rule class defines an *Event-Condition-Action* (ECA) rule, being triggered by an Event that produces the execution of an Action. These Rules define particular interconnections between instances of the Event and Action classes; including the configuration parameters for both: inputs and outputs.

2.2.5 Android App

A Mobile Application (developed by Antonio Fernández Llamas [7]) is also used in this project so the custom content can be presented to the users. The mobile interacts with the Beacons through Bluetooth, processing data when an event is captured and a rule is then evaluated.

2.3 Custom Content Management System

Having already gone over the technologies that enable the delivery of the contents in this project, we will focus now on the ones enabling the creation and storage of such content. The project should allow the creation of customized content so we can target the highest number of users depending on the context. It should be easy, accessible and reliable.

There are several alternatives available on the Internet when it comes to HTML Content editors. We have taken a few of them into account: *CKEditor[17]* or *TinyMCE[5]* being good examples of "What You See Is What You Get" editors. The thing is that this kind of editors share all the same features. And more often than not these are bloated with a lot of additional tools that are not used, especially in the context of this project.

We also looked into already prepared solutions for content management in Beacon environments, such as *Pushmote[9]*. The main issue with these solutions is that they are strictly closed. Meaning that you can only use what they allow, so there is no margin to extend it.

2.3.1 SirTrevor

SirTrevor [13] is an open-source javascript library that provides a rich content editor, and not just for the web. Instead of storing the content in a database, SirTrevor stores it inside a JSON object. It also provides a very clean and simple interface that allows us to create custom content in an easy way.

What makes SirTrevor different is that its purpose is to create structured content with no presentational information. It is often called "output-agnostic" – meaning that how the content is rendered is up to who uses it. What really matters is the content and the structure, but not the presentation.

Being open-source, lightweight and extensible, SirTrevor seemed like the best choice.

2.3.1.1 Block Architecture

SirTrevor content is made out of **blocks**. A block defines the type of content and the actual data inside that block. Sir Trevor includes a collection of pre-built blocks that you can include and use inmediately, or use as a base for your own custom blocks. In our case, these are the blocks that we have included in the project: Text, Heading, List, Image, Tweet and Video.

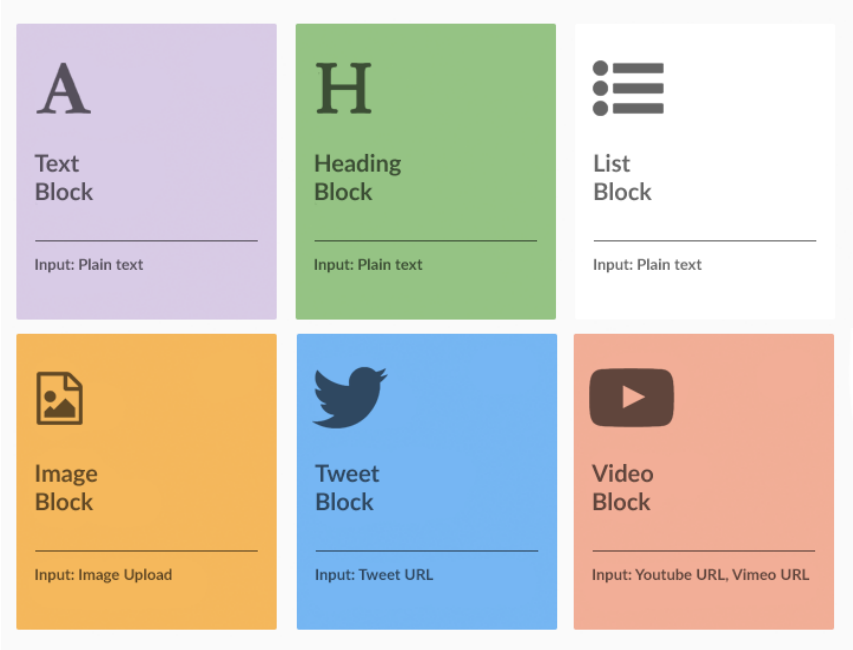

Figure 2.4: SirTrevor Block Catalog

- **Text Block:** Allows both plain and rich text (including hyperlinks, bold and italics) to be structured in a simple way.
- **Heading Block:** Useful for design purposes and creating sections in your content.
- **List Block:** Unordered lists are also available as a block.

- **Image Block:** Uploading your own images is also an option with SirTrevor (more on how to handle the upload in the Architecture Section of this project)
- **Tweet Block:** Useful way to embed single tweets.
- **Video Block:** Embedding videos from popular video services on the Internet is also a possibility.

It is very important to remember that SirTrevor does not store any of this, it only creates a JSON object that remembers the order of the blocks and its contents.

All of these blocks can be easily translated into HTML (more on this later on the Architecture section), and they all follow the next structure:

Listing 2.1: Text Block Structure

```
{
  "data": [{
    "type": "text",
    "data": {
      "text": "<p>Hello, my name is <b>Sir Trevor</b></p>",
      "format": "html"
    }
  }]
}
```

The block structure is always composed of both *"type"* and *"data"*.

That is why in order to create our Content Management System we need to store this object in a database. We have opted for a NoSQL database, MongoDB [14], as explained later in this section.

2.3.2 Apache Stanbol

Apache Stanbol [1] is an open source set of software components for semantic content management. Its intended use is to extend a content management system with semantic services. In order to integrate and be used as a semantic service for a CMS, all of its components can be accessed via a RESTful web service API. Its main features are:

- **Content Enhancement:** Extracting data from non-semantic content in order to add semantic information.

Figure 2.5: Apache Stanbol Functionality

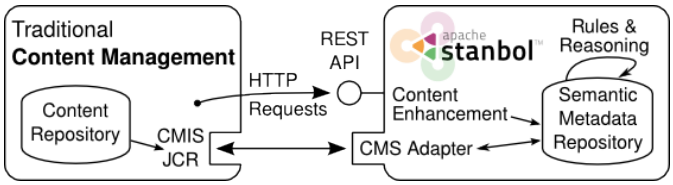

Figure 2.6: CMS using Apache Stanbol via HTTP Rest Interface

- **Reasoning:** Services that are able to retrieve additional semantic information from the enhanced content described before.
- **Knowledge Models:** Definition and manipulation of the data models (ontologies) that are used to store the semantic content.
- **Persistence:** Storing and/or caching of semantic information, so it allows search.

Apache Stanbol is a really interesting project, although its learning curve is very steep. It also may turn out a bit *overkill* for our project, since we are not focusing the custom content to a specific model. Despite not going ahead with it for this project, it will be considered for future lines of work.

2.3.3 MongoDB

MongoDB[14] is an open-source cross-platform database that stores data using a flexible document data model very similar to JSON. The fields inside can vary from one entry to another. Instead of being based of tables and rows like in relational databases (such as SQL), MongoDB is built of documents and collections. Documents structure are comprised of sets of key-value pairs. Collections are simply sets of documents, being the equivalent to tables on the traditional relational databases.

The major asset of MongoDB, and other NoSQL databases, is how dynamic documents can be. Documents contain one or more fields, including arrays, binary data or sub-documents. And the structure can vary from one document to another with no trouble at all.This allows to evolve the data model as quick as we need it to change.

MongoDB uses BSON (Binary JSON) behind the scenes. BSON extends the JSON model providing new features like additional data types and the possibility of ordering fields. Given that SirTrevor stores the data in a JSON object, MongoDB seemed like the perfect choice for our Content Management System.

Conclusion

Having considered the different approaches and technologies over the last pages, we have opted for a combination of different technologies for this project.

We will create our own Custom Content Management System, or CMS, using SirTrevor, MongoDB and web technologies such as PHP and Javascript. The CMS will be then integrated into the EWE-Tasker web application and share the same user database.

The content will be served to the Android Mobile Application depending on the user context and the semantic data of the contents. The mobile application will be modified in order to fulfill this purpose.

CHAPTER 3

Architecture

In this chapter, we will explain the architecture of this project, including the design phase and implementation details. First of all, different content management system alternatives will be evaluated. In the second place, we will focus on the architecture of the project itself. Finally, we will talk about the integration on both the EWETasker web-app and the Android app.

3.1 Alternative Beacon CMS Solutions

Since we decided to go ahead and use the Estimote Beacons, it is worth mentioning that the company had plans for building a project like ours. In a 2013 interview [16] with Estimote CEO Jakub Krzych, he talks about a cloud based Context Management System. It could be used to associate or define custom messages or images when a mobile device enters a Beacon region.

Since then Estimote decided not to pursue the idea and focus instead on providing the best solutions for the actual beacons. Improving the hardware (different sizes and types of beacons for different solutions), the firmware (support for iBeacon, Eddystone and Nearable), the mobile SDKs, indoor positioning, security, etc.

In the previous chapter we briefly described different solutions in order to present custom content to the users on a beacon environment. These may be suitable in some situations, so we have decided to evaluate a few and see if they could work for our project.

Pushmote

Pushmote [9] is a Software Development Kit, available for both Android and iOS, that allows your app to show content to your users depending on their location.

Similar to the way the Task Automation Server works, Pushmote defines three key terms to understand the way it works.

- **Scenario:** A Scenario is a sentence relating an *event* and an *action*. Its the Pushmote equivalent for a Rule in our Task Automation Server.
- **Event:** Possible events (all location related) are: **Enters to**, **Exit from** and **Comes nearby to**.
- **Action:** These may include: Showing an image, a video or opening an URL.

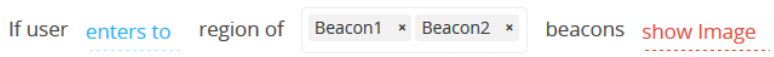

Figure 3.1: Pushmote Scenario, showing the Event and the Action

There are some disadvantages with solutions like Pushmote, we will go over these now.

First of all, there is no context present on it. The events we can use on a scenario are exclusively location based, so there is no customization on what kind of content a user will receive. If two different people enter the same region they both will receive the same content.

Also there is so much you can do with it. The actions are not extensible, you can only use the ones that Pushmote defines. There is no way to build your own solution and integrate it.

Lastly, they do not offer free plans. In order to try it we have had to make use of a limited trial. We would rather go with an open-source solution.

Rover

Rover [12] is another company focused on location-powered mobile engagement. Even though they have been on the market less time than other competitors, they really stand out from them.

The main features are very similar to the ones that Pushmote offers (location based events, beacon management). The thing that makes Rover stand out from others, is that it adds a bit of Context into their solution. On top of location-triggered events, you can add more filters, based on the customer age or gender for example. Rover also provides a *Experience Creator*, a great custom content creator that allows you to design the content straight from their website, customizing everything.

The drawback is that Rover, having all these features, is not free. They offer too a 14-day trial, but their cheapest plan starts from $150 / month. Despite that, they are a great inspiration for this project.

Bleesk

Bleesk [2] is the last of the companies that we have evaluated. The main focus of Bleesk is simple: download their app and start receiving the best local offers and deals.

Their approach is a bit different from the other ones. They offer what is called *White Label*. They rebrand both the mobile app and the CMS so you can offer it under your brand to your customers. Although it is an interesting idea, it is not suitable for our project.

Conclusion

After evaluating the different alternatives we still think the best approach is creating our own system. None of the evaluated solutions have everything we are aiming for, but we will be using them as inspiration.

We will design and implement a content management system to manage contents through a web application. In addition, we will extend it by adding a model using ontologies, getting the semantic context functionality we are looking for. This will allow us to serve content depending on the context, and even automatically recommend custom content based on semantic technologies.

3.2 Architecture Overview

In this chapter, the architecture of the project will be explained, including the design and the final implementation. The system is divided into different modules to help its understanding.

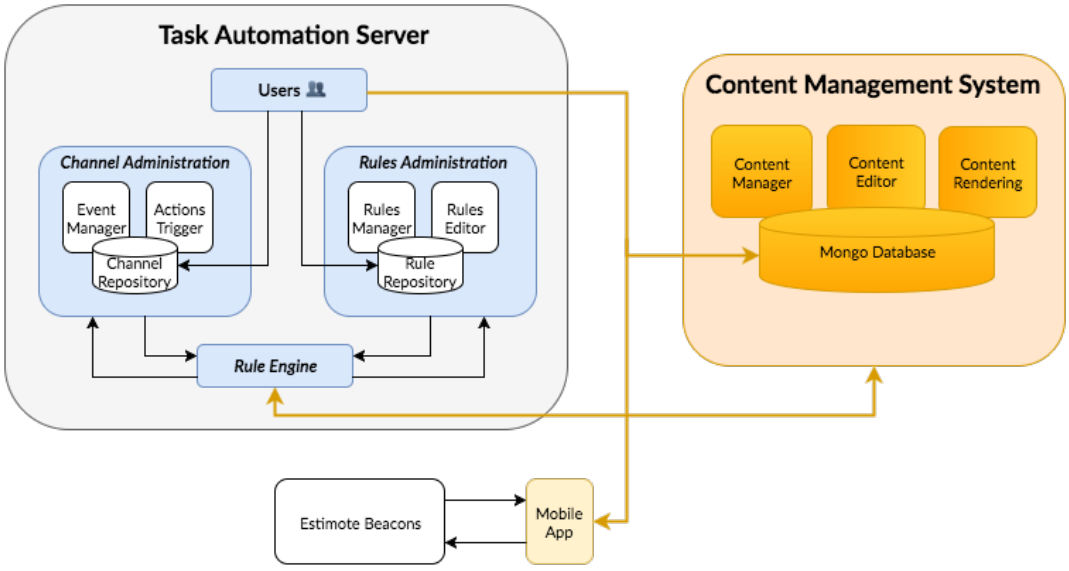

Figure 3.2: Architecture

The figure 3.1. shows the whole system, which is mainly composed by the Content Management System, the Task Automation Server and the Mobile Application.

- **Content Management System (CMS):** this module is the responsible of creating and managing custom content destined to the users. This module is divided into three

different submodules:

  - Content Manager
  - Content Editor
  - Content Rendering

- **Task Automation Server:** this module [15] is the one responsible of automating the content being delivered to the users making use of rules.

- **Mobile Application:** The Android application [7] is where the content will be delivered and presented to the user when certain conditions are met.

Events coming from the Estimote beacons are passed to the Task Automation Server via the Mobile App. Once these events are received an action is triggered, so the CMS receives the command and executes an action. The content is then delivered to the mobile app where it will be shown.

The main focus of our project is the Content Management System, and in a minor way the Task Automation Server and the Mobile App.

In the following sections we are going to describe deeply subsystems involved in the project.

3.3 Content Management System (CMS)

One of the main uses of beacons is to provide contextual information about the object the user is next to. This is not exclusively restricted to objects, and can also apply to places so information is provided about the environment. Other use cases may be: authorizing access to rooms or buildings, navigation help in specific areas, enhancing accessibility, or insightful data about the people entering and exiting the place.

For this purpose we have developed a Content Management System allowing creation and management of the content that will be presented to the users in many of these cases.

The CMS has been implemented using different technologies like HTML5, CSS3, Javascript, MongoDB and PHP.

3.3.1 Content Editor

Technology Stack

PHP is a server-side scripting language originally created in 1994, especially suited for web development and can be embedded into HTML.

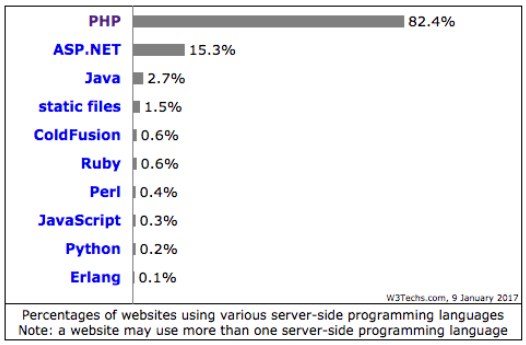

Figure 3.3: PHP Market Share in 2017

PHP is used by 82.4% of all of the websites using server-side scripting. Despite the down on the popularity that it suffered the past few years, with the new versions it has rised again. PHP is a solid choice for a project like ours.

Also, as we have stated before we decided to incorporate a Javascript library into our CMS. SirTrevor is a pretty straight-forward out-of-the-box library, and the documentation is more than enough to get a project started really fast.

There is only two steps to get SirTrevor up and running: first you include the library files into your project, and last you incorporate a SirTrevor element inside a form in your HTML that will get converted to a SirTrevor instance with a small Javascript snippet.

```
<form>
  <textarea class="js-st-instance"></textarea>
</form>
```

```
<script>
  var editor = new SirTrevor.Editor({
    el: document.querySelector('.js-st-instance'),
    defaultType: 'Text',
    iconUrl: 'build/sir-trevor-icons.svg'
  });
</script>
```

Going back to the state of the art, we talked about how SirTrevor stores the content you

create. It is all contained in a JSON file. SirTrevor does not provide a solution for storing the content, so we decided to use MongoDB to store the output of the editor.

Adding an intuitive interface, some design principles and the power of Javascript and CSS, we present the result for the graphic interface of our Content Editor.

Graphic Interface

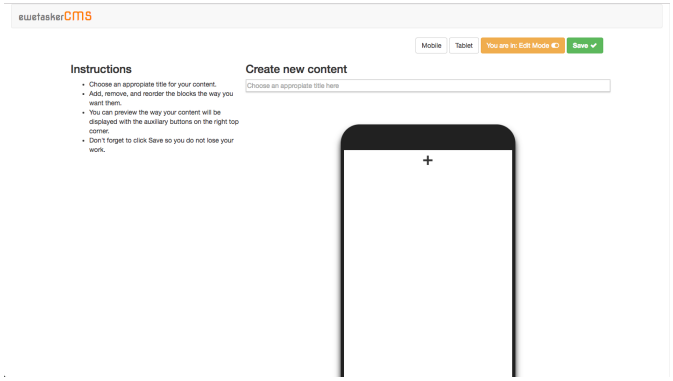

Figure 3.4: Content Editor

The Figure 3.4. shows the main interface of the Content Editor. With some custom CSS and Javascript code we have made the SirTrevor instance into an actual smartphone to make the interaction better. Basic instructions are also displayed on the left side of it, and a title textbox is placed on top of the preview area.

An interesting feature added into the editor is the possibility of previewing the way your content will be displayed depending on what kind of smart device opens it. The control buttons are located on the top right corner.

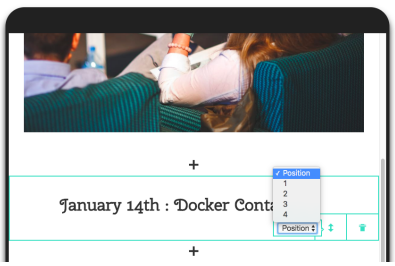

Figure 3.5: Content Editor: Block Management

SirTrevor main features on block management are available. You can add, edit, delete or even modify the order your content blocks are displayed. At the same time you are doing the changes, the JSON file is dinamically changing to reflect all these modifications.

We have also developed a listing interface in order to manage all the contents from the database. In the first stage of the project development, all contents are shown regardless of who created them since the user integration came at a later stage. From this view we can edit content that is already created, we can render it as HTML (the way it will actually appear on the device, more on this on the section "Content Rendering") and we can also delete items. A shortcut button to create new content has also been added on the top right corner.

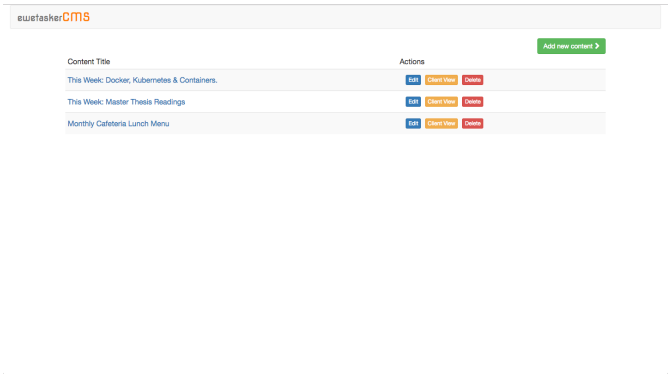

Figure 3.6: Content Editor: Content List

3rd Party APIs (Image/Tweet Block)

All of the default SirTrevor blocks work with no customization needed at all. There are two exceptions to this rule, and these two are the Image and Tweet block.

Image Block: Image Upload API

The Image block relies on a server side component to store images on the server. The default behavior of SirTrevor will be doing an AJAX file upload in the background to a server endpoint. It will then retrieve the image URL, hosted on our own server.

In order to not be server dependent in our project and avoid handling image files, we decided to incorporate a third-party API as a solution for this problem. Images will be hosted on an external server.

Imgur is an online image host and image sharing community founded by Alan Schaaf on 2009. It was created as a response to the standard problems encountered in the image hosting services. Started as a side project, it has raised $40 millions in funding and nowadays is the first choice for many when it comes to image uploading.

In order to use Imgur services from the RESTful API an authorization is needed so our

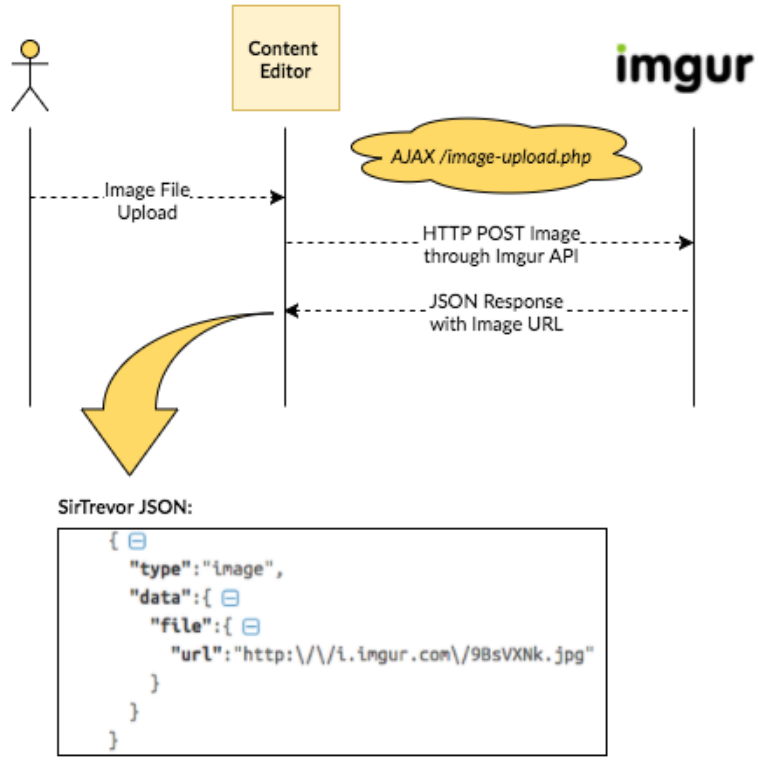

Figure 3.7: Image Upload Processt

application must be registered in Imgur as a developer application. After registering the application we will obtain a Client-ID, that allows us to upload images to Imgur servers (and store them into our account).

It is important to remember that every HTTP request we make to the Imgur API will need to incorporate the next HTTP header: *"Authorization: Client-ID ¡our-client-id-here¿"*.

The only remaining thing to do was to code our custom server-side solution on PHP. After selecting an image from the file upload we need to encode the image on base64 and send it in a POST to Imgur.

This is an excerpt of the JSON response received after an image upload:

```
{
  "data": {
    "id": "QEhi1py",
    "title": null,
    "description": null,
    "datetime": 1484007293,
    "type": "image/jpeg",
    "animated": false,
```

```
    "width": 960,
    "height": 638,
    "size": 64580,
    "link": "http://i.imgur.com/QEhi1py.jpg"
  },
  "success": true,
  "status": 200
}
```


The important part for our Image Block is the actual link of the image. So we will return that in our PHP script, and our SirTrevor instance on the Content Editor will pick it up and show the image successfully.

Tweet Block: Twitter API

Similar to the Image Block, the Tweet Block needs server-side code. The code will need to lookup a Tweet ID that will be introduced in the block in the Content Editor, and it will return the tweet in JSON making use of the official Twitter API.

Unlike the Imgur API, the only way to do this involves making use of an authenticated call. More specifically, Twitter uses OAuth to provide authorized access to its RESTful API.

Open Authorization, more commonly known as OAuth is an open standard for token-based authentication and authorization on the Internet. It allows an end user's account information to be used by third-party services (in this case, Twitter) without having to share the user password.

Since we do not really need to do things like posting tweets, meaning a user context, Twitter provides Application-only authentication. Its implementation is based on the Client Credentials Grant flow of the OAuth 2 Specification.

Since there is no need to sign a request, this approach is much simpler than the standard OAuth model.

After getting our access token we just need to make use of the following endpoint: *GET https://api.twitter.com/1.1/statuses/show.json?id=tweetID*

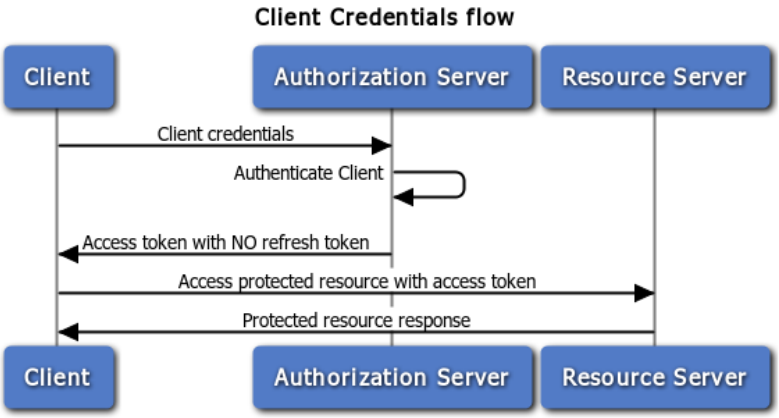

Figure 3.8: Client Credentials Grant Flow, OAuth 2.0

3.3.2 Semantic Information: Place Ontology

We wanted to associate the content to a specific location, in order to let the Content Delivery System send the appropriate content. With this purpose in mind, we have defined a Place Ontology. Making use of available standardized vocabularies we have created the following, using the EWE ontology:

Listing 3.1: Place EWE Ontology

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix dbpedia: <http://dbpedia.org/resource/classes#> .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/> .
@prefix ewe-place: <http://gsi.dit.upm.es/ontologies/ewe-place/ns/> .
@base  <http://gsi.dit.upm.es/ontologies/ewe-place/ns/> .

ewe-place:Place a owl:Class ;
  dc:title
  dc:description
  geo:lat
  geo:long
  dbpedia:floorCount
```

- **dc:title** - This represents an appropiate title for the location.
- **dc:description** - A brief description of the place we are defining.
- **geo:lat geo:long** - Latitude and Longitude pointing to the location.

- **dbpedia:floorCount** - In a building context, it is useful to define the floor in which the place is located.

The content created is then associated to the place we want to, and all of this is stored in the Content Management System database. The Content Delivery System will then, after getting an event that a user is in a smart place, retrieve the corresponding content to the area.

3.3.3 Content Rendering

The content is going to be rendered on mobile devices. When it comes to content rendering (meaning text and media) the different operating systems have notable differences. This is reason enough for a lot of application developers to resort to web technologies.

Web technologies on mobile are a great and solid choice. Content rendering in different operating systems is almost identical in the eyes of the user, and it makes the job easier for the developers. You can even have webviews embedded on your applications without the user even noticing.

Responsive web is a must nowadays, and we have reached a point where web content is more consumed on mobile devices than on desktop.

But, our Content Management System is "output-agnostic", meaning that how the content is rendered is up to who retrieves it. This is the reason we have built a module following a common design pattern, an Adapter.

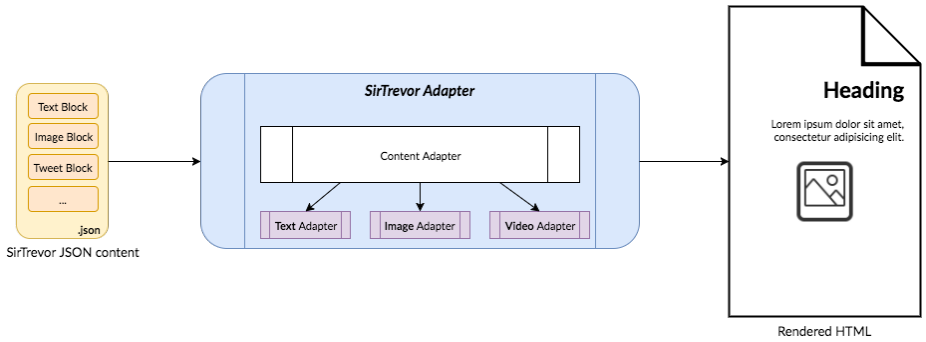

Figure 3.9: CMS: SirTrevor Adapter

The adapter is built on PHP. Using MongoDB controller, the adapter retrieves the requested content. The adapter loops through the JSON object, iterating over all the blocks. Depending on the type of the block, it will call the corresponding static sub-module. The input will be the block data, and the output is HTML.

After the adapter is done we have a complete web-page containing all the content ready to view from the mobile device.

3.4 Content Delivery

We have already gone through the main module of our project. In this section we will talk about how the content is delivered to the users. This task is accomplished by integrating our developed Content Management System into the Task Automation Server [15].

3.4.1 Task Automation Server

The main purpose of this module [15] , designed by the Intelligent Systems Group, is to manage certain tasks handling events and triggering actions in an automated way. This module is composed by four modules:

- Rule Engine: this submodule evaluates user created rules. If the conditions are met an action triggers.
- Action Trigger: submodule that triggers an action if it receives the appropriate response.
- Rule Administration: this submodule allows users to create new rules and manage existing ones.
- Channel Administration: submodule

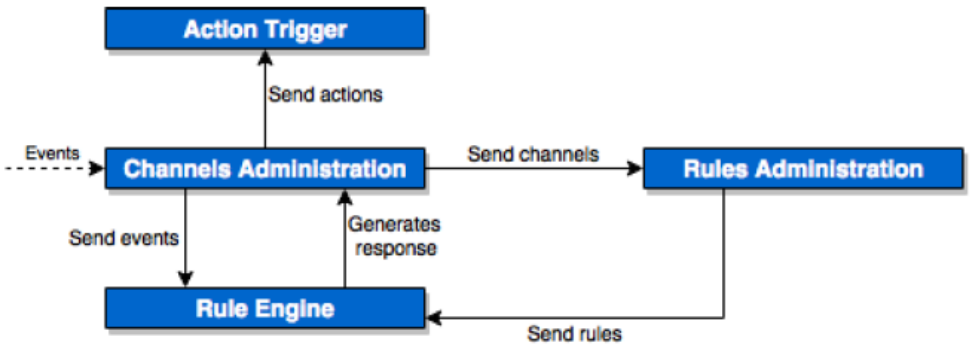

Figure 3.10: TAS Sub-modules Interconnection

Rule Engine

Rule Engine is based on the EWE ontology we have described on the Enabling Technologies section of this project. It is composed by the EYE Helper and the EYE Server.

The EYE Helper loads the user rules. It then captures the events and sends them to the EYE Server.

The EYE Server evaluates the received events and rules, and draws conclusions from them. These are the actions, that will be then triggered by the Action Trigger submodule.

Rule Administration

The submodule allows creation and management of user rules. It is composed by the Rule Editor, Rule Manager and Rule Repository.

Rules have a Event - Condition - Action structure, commonly known as "If this then that".

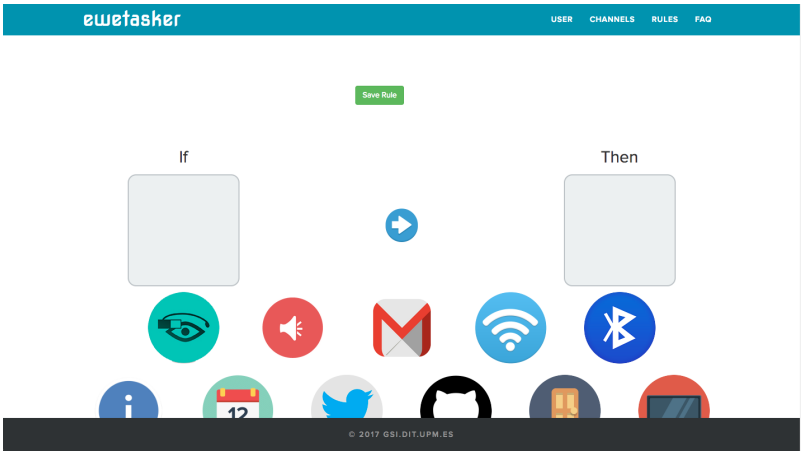

Figure 3.11: Rule Editor Graphic Interface

New rules have been defined for this project in order to automate tasks related to content delivery and location. The rules are the following:

- If the user presence is detected near a specific beacon, the user has entered a Smart Place.
- If the user is inside of a Smart Place, custom content is delivered to the user mobile device.

Channel Administration

This submodule allows new channel creation and management of existing ones. It also handles the events and the parsing of the response generated when a rule is evaluated. It is composed by: Channel Editor, Channel Manager, Channel Repository and Events Manager.

Channels are defined by the events and actions associated to them, using the EWE ontology. These channels are stored in the Channel Repository using JSON, and the structure follows the next schema:

Listing 3.2: New Place Channel

```
{
  "title": "Place",
  "description": "This channel represents a specific place powered by
      beacons.",
  "nicename": "Place",
  "created_by": "jav",
  "events": {
    "event": {
      "title": "Inside Of",
      "prefix": "@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
           . @prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#>.
          @prefix ewe-place: <http://gsi.dit.upm.es/ontologies/ewe-place/ns
          /#> .",
      "num_of_params": "1",
      "eye_fragment": "?event rdf:type ewe-place:Inside.
      ?event ewe:placeID ?placeID."
    }
  },
  "actions:": {
    action":{
      "title": "Entered",
      "prefix": "@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
           . @prefix ewe-place: <http://gsi.dit.upm.es/ontologies/ewe-place
          /ns/#> . @prefix ov: <http://vocab.org/open/#> .",
      "num_of_params": "1",
      "eye_fragment": "ewe-place:Place rdf:type ewe-place:Inside ;
      ov:location "#PARAM_1#"."
    }
  },
  "created_at":"12:13 03-01-2017"
}
```

Two new channels have been defined in this project, allowing us to accomplish our objectives and leaving room for new ways to use them.

Place Channel

This channel represents a specific place in a smart beacon environment. Having a Place channel is really useful so users can automate tasks when they enter specific places, avoiding having to deal with the Beacon IDs and using a more natural concept.

- Event: Inside Of. When a user is inside of a specific place.
  - Params: Identifier of Place
- Action: Entered. Informs that a user has entered the place.
  - Params: Location

Having the Place channel allows users to automate any tasks they want within a specific place. Users will have a selection of smart places available to choose from.

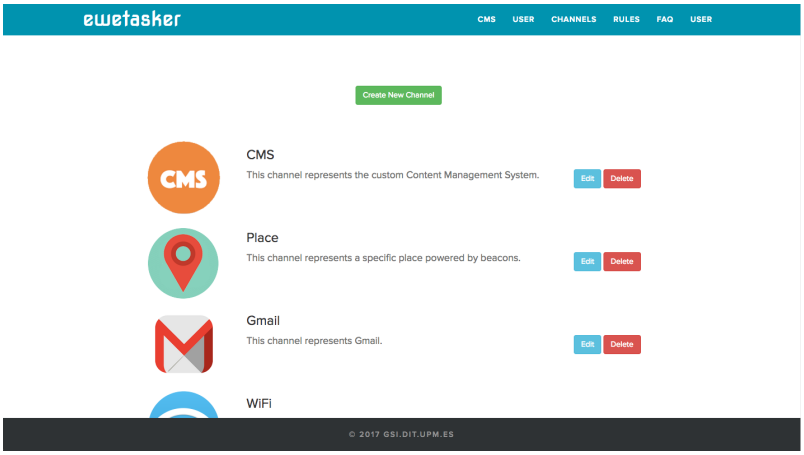

Figure 3.12: Channel Administration Interface

CMS Channel

This channel represents the custom Content Management System already described in the last section. This channel is essential so automated appropiate content is shown to the users, so we need to define an action for this.

- Action: Show. Shows appropriate content depending on the context.
  - Params: Location

3.4.2 CMS Integration

We have described in the last section the different submodules that compose the Task Automation Server. We decided to integrate the CMS into the EWE-Tasker web application so we could make use of the same database.

Now the users can create their custom content using the same login that they have in order to create rules.

The CMS Channel action to show content needs to be executed when triggered. In order to do so, it will make use of an API developed in the CMS module. HTTP requests will be sent to the CMS asking for the appropiate URL that will get delivered to the mobile device.

Once the URL is delivered to the mobile phone the user needs to be notified. When they click on the notification the content will appear on their screen.

In the final section of this chapter we will talk about how the content is finally rendered on the mobile device using an Android application.

3.5 Android Application

A Mobile Application is used in this project to show the content to the users. We decided to take the existing application [7] developed by the group and improve it in order to accomplish our goals.

The mobile application allows us to interact with the beacons located on the smart places. The beacons are constantly broadcasting via Bluetooth, letting smart devices listening they are close.

Once the event of being close to one of the beacons is captured, the corresponding rules will be evaluated. Therefore, locating the user in the Place and sending available content to their device.

When the content URL is sent to the mobile application, a notification comes through the phone alerting the user.

If the user clicks on this notification, the application opens with the custom content already rendered.

CHAPTER 4

Case study

Introduction

This chapter objective is to help understand the project main functionalities. For this purpose we will go over the main use case to show the main features.

We want our users to automate tasks in their work environment making use of the new channels we have created, Place and CMS. Making users more comfortable in the workplace will result in a productivity increase, and a better mood. Also, we could make time-wasting tasks easier and remove dependencies, like in our main use case.

4.1 Enhanced Workplace Navigation

This use case is aimed to make the adaptation curve easier when someone starts a new job. For this scenario we will be using the Intelligent Systems Group as the Company. The laboratory of the Group will act as the workplace. This laboratory is located inside our School, in the Technical University Campus on Madrid.

First, we have to identify the main actors in this use case:

- **Administrator User:** the goal of this user is to configure the different modules so the automated smart environment is set correctly for the other user. This way the company can ensure that the new candidates are properly introduced to the workplace.
- **New User:** this user goal is to experience a nice transition into his new work environment. Using the mobile application the user will encounter convenient features, learning more about his surroundings, making his transition to the new workplace easier.

In order to build the scenario for this use case, we have used the available research done in the School Ambassadors program [8]. This program purpose was to elaborate an appropriate script for guided routes, during an open house day on the Telecommunications School. This guided visit was oriented to high school students that were considering joining the Technical University after graduatin g.

Different routes were designed in order to target the different research areas that the school offers. Considering the Intelligent Systems Group we have adapted Route A and made it more suitable to our needs.

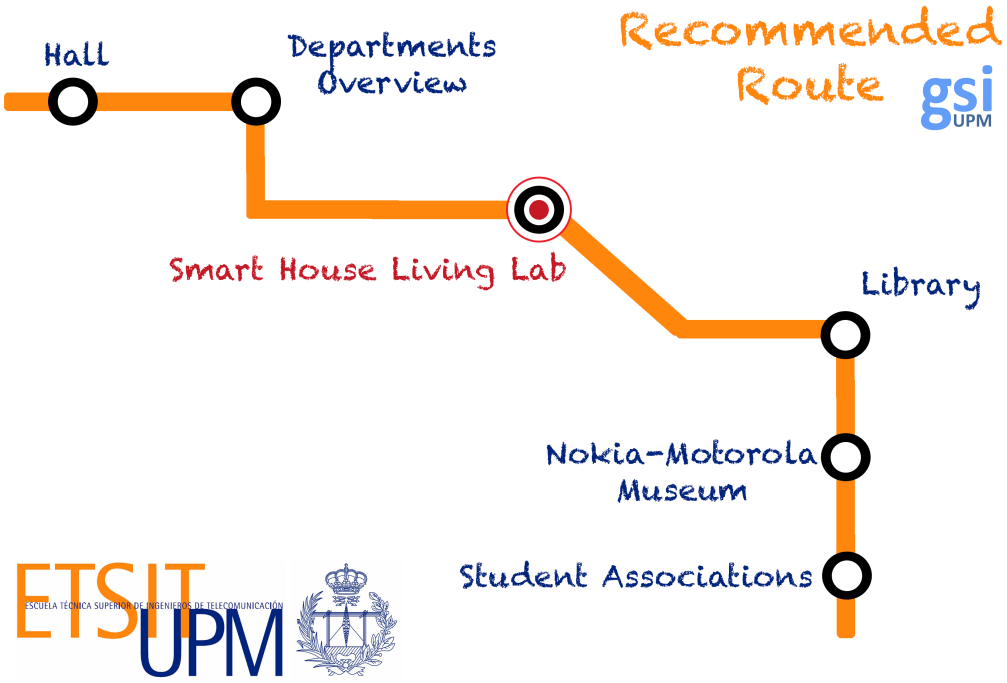

Figure 4.1: Case Use: Enhanced Navigation Route

Coordinating an open house day is a tedious work, involving many people in order to

show people around. This is affordable in terms of effort when you have dozens of people attending. If only one person is joining the team there is no easy way to show them around.

Using the research previously mentioned we have enough information to create custom content for each place. And that is the next step, creating the custom content for each location.

Content Creation

Before being allowed to access the Content Editor, the administrator will have to sign into the system. As we mentioned on the Architecture chapter, the CMS was integrated into EWE-Tasker to make things easier. So, by using the same login credentials we can access both functions. All users can access the CMS, it is not restricted to just administrators.

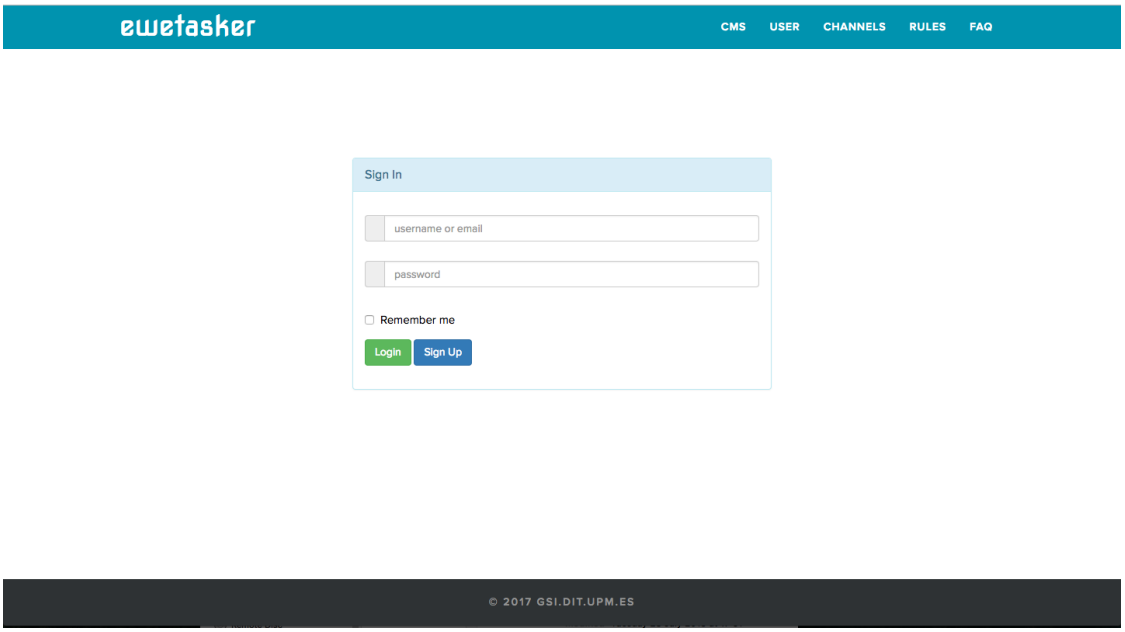

Figure 4.2: CMS: Login form

In this use case, the administrator will be using the CMS for the first time. So when he access the CMS he will find that there is no content available, but he will be prompted to create it.

After clicking on the “Add new content“ button, the administrator will start creating the custom content. Adding text, images, even videos is easy using a clickable and intuitive interface.

There is a textarea where the user will use the designed EWE Place Ontology described

in the Architecture chapter. In this case, our administrator is creating the content that will be associated to the School Library.

Listing 4.1: EWE-Place Ontology describing the School Library

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/> .
@prefix ewe-place: <http://gsi.dit.upm.es/ontologies/ewe-place/ns/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix dbpedia: <http://dbpedia.org/resource/classes#> .
@base  <http://gsi.dit.upm.es/ontologies/ewe-place/ns/> .

ewe-place:Place a owl:Class ;
  dc:title "Library";
  dc:description "The library is located on the A building.";
  geo:lat "40.452014";
  geo:long "-3.725808";
  dbpedia:floorCount "1" .
```

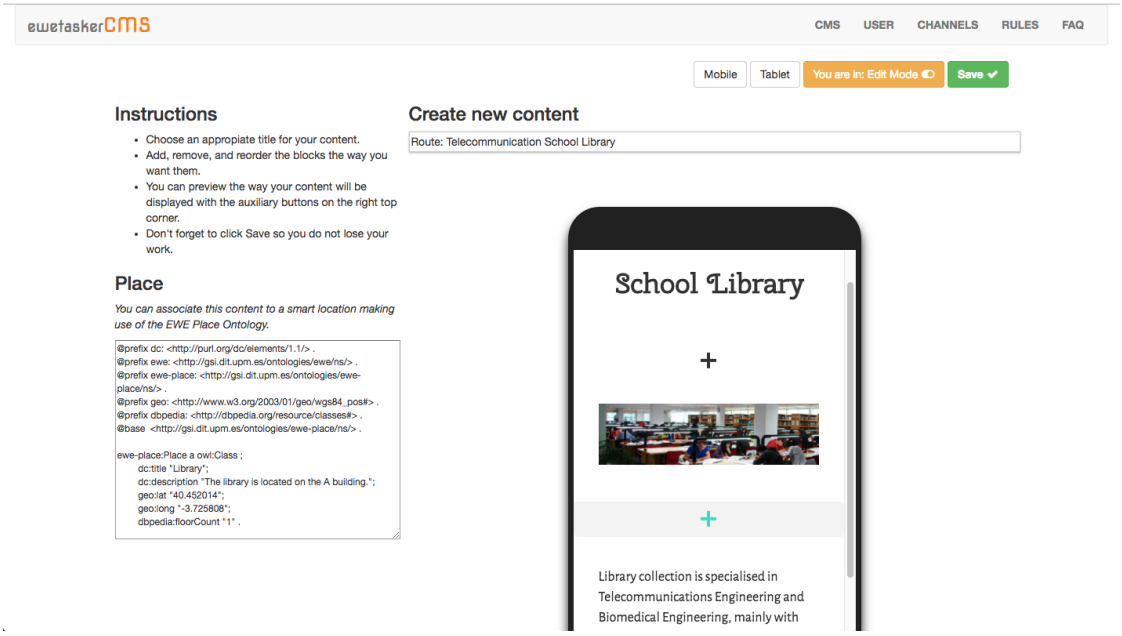

Figure 4.3: CMS: Creating Custom Content

After the administrator is satisfied with the output, he will click Save in order to store it on the Content Database. He will continue to do the same procedure with all the different content he needs to create in order to finish the route.

It is advised to create a user account with the only goal of managin all this content.

That way the edition of the content over time will be easier, not being dependable on one single person, and less an administrator.

In the following figure we can see the list of contents already created by the administrator user. Each one of these contents has a "Place" associated using the EWE Place Ontology. The user can manage this content as much as he wants. Different options are given for every content:

- Edit: clicking this button allows users to modify the existing content on the Content Editor.
- Client View: this is a full-page preview on how the content will be rendered as HTML. Using developer tools from any of the current browsers we will be able to replicate any screen size. It is really useful in terms of QA and testing.
- Delete: clicking this button, and after confirmation, users will be able to delete existing content.

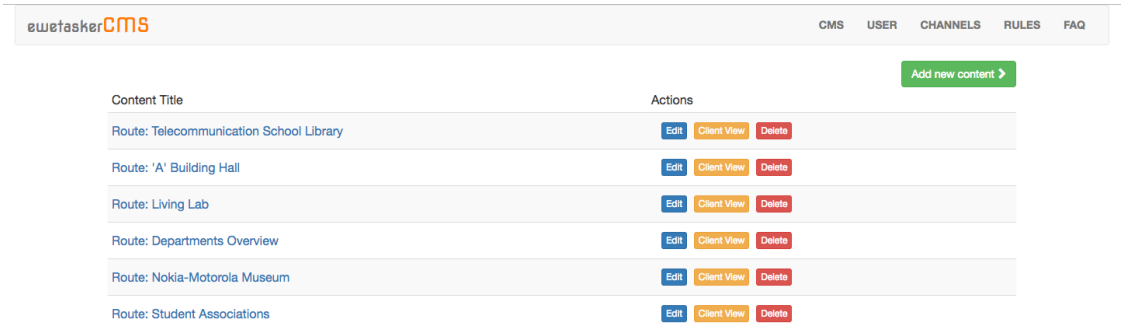

Figure 4.4: CMS: Content Listing

EWE-Tasker Configuration

We have already prepared all the content that will be displayed on the different locations. It is time to configure EWE-Tasker in order to automate the content delivery for the users.

Channel Creation

The first step involves Channel creation. Once a channel is created, it will be available to any user on the database so they can create custom rules as they please. For this project we needed to define two new channels: Place and CMS.

In order to do so we click on Channels in the top-menu bar. We will encounter a list of all the available channels, and a button allowing us to create a new one. Clicking in said button we will encounter a form.

Every channel has common fields: the title, the description, a nice name, and an image to represent the channel. The image is needed in order to create rules, as we will see later in this chapter.

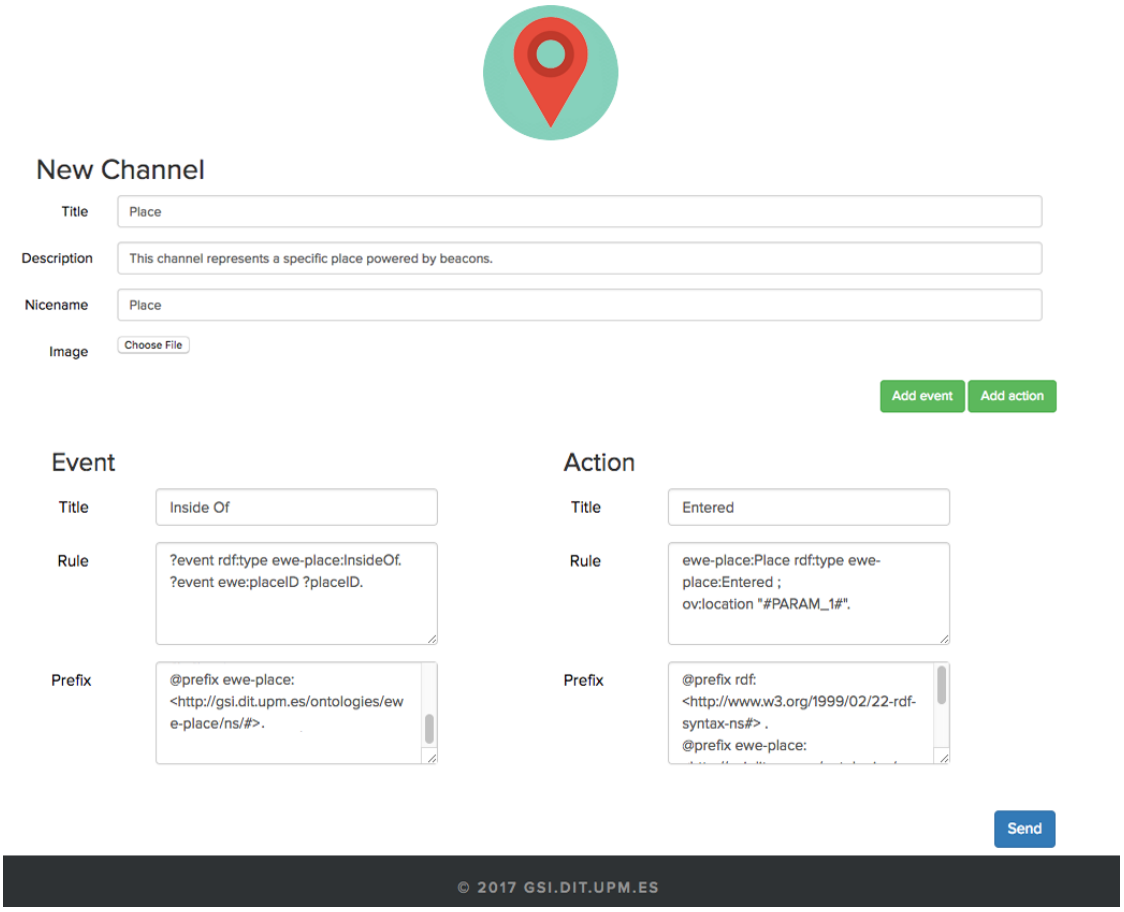

Figure 4.5: EWE-Tasker: Place Channel Creation

We can also add different events and actions, as it was described on previous chapters. When the channel has everything we need, the Send button is clicked and the channel gets stored in the database. The channel is now available to create rules on the EWE-Tasker platform. We also need to do this with the CMS channel. Both channels are described in appendix A.

Rule Creation

Once we have the Place and CMS channels in our database, it is time to create the rules. These rules will be automatically executed, achieving our ultimate goal: displaying content on the user device.

First, we need to define the rules that will associate a Place with a Beacon. Defining these rules we are allowing users of the EWE-Tasker platform to create their own rules based on location in an easy way.

The first two params of the rule come from the event of the Presence Detector Channel. The first one being the beacon identificator, and the second one the distance to it. The last param of the rule indicates the Place where this beacon will be placed.

This rule will be replicated with different data for every smart place we want to enable.

In order to show content we now need to capture the event of being in a Place. We do not need to define a rule per location, as we did before.

The new rule just has to be defined once, and it will be imported into the new user account so it can be executed, along with the previously defined rules.

Figure 4.6: EWE-Tasker: Created Rule

Content Rendering

Finally, when all of the previous one-time setup has been completed, the user is ready to receive the content on his phone.

To do so, the user has to install the already introduced Mobile App [7] loggin with his user account, in order to listen to the beacons present in the workplace.

Whenever the user passes through a beacon located on the entrance of the Library, the event will be sent to the Task Automation Server, executing the corresponding action to locate the user on the Library.

When the mobile phone receives the confirmation of being located on the library from the TAS Server, it will trigger the API call to the CMS server, retrieving the appropiate content.

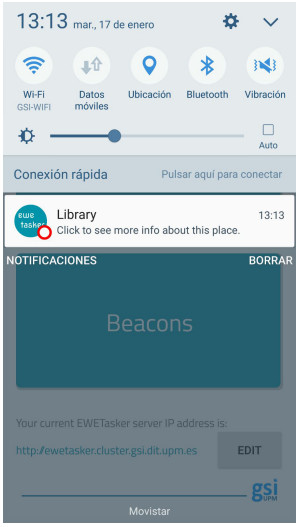

Figure 4.7: Smart Place Notification

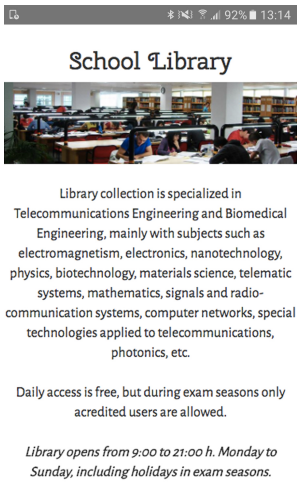

Figure 4.8: Displaying Content

A notification will be displayed on the user screen alerting him that there is information about this location available. Tapping on the notification will render the content.

4.2 Other Use Cases

The main use case described in the last section provides insight about how the project functionalities may be of use, displaying the main features.

There are plenty of cases where this project can be useful, we will list a few of them as a conclusion for the Case Study chapter.

Weekly Workshops and Talks Newsletter

Employees get a brief introductory text and a video about the weekly talk depending on their interests, previously declared on their account. This could also apply to other company events or announcements.

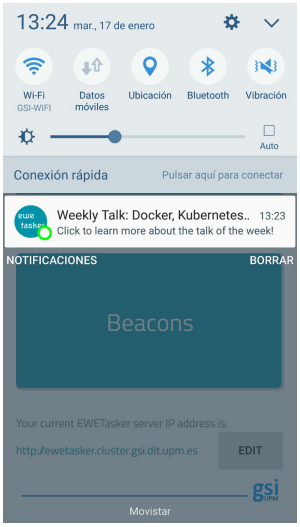

Figure 4.9: Weekly Talk Notification

Figure 4.10: Displaying Content

Emergency Evacuation Guidance

By triggering an emergency flag, the system will send users an emergency alert notification. The custom content could show the optimal evacuation routes depending where the users are located in that moment.

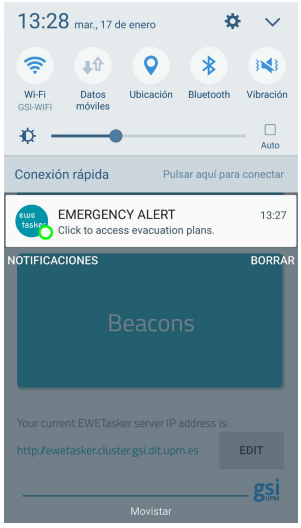

Figure 4.11: Mobile App: Emergency Alert

All of these cases are easily achievable through the edition of the Content Management System API. Having access to user profiles makes the targetting of the content more accurate.

CHAPTER 5

Conclusions and future work

5.1 Introduction

In this last chapter and to conclude this project, we will describe the conclusions obtained with the project completion. We will resume the principal concepts explained in the memory, the achieved goals, and finally a brief discussion about future work for the long road of this project.

5.2 Conclusions

Having finished this project, and through the use of new technologies such as beacons or semantics, we have proven that context-aware contents can improve the way users interact with their work environment.

We have developed a Custom Content Management System, allowing users to create content that may include text, images, videos and even tweets. Users can also associate this content to different locations with the help of semantic notation.

We have also integrated this CMS into a Task Automation Server, so their users can

access it and content delivery gets automated through rule creation.

In this context, we have learned that using semantic technologies and automation can help in the process of content customizing and targeting.

Lastly, we have adapted a mobile Android application so the users can receive the content created in the CMS depending on their context.

The project has been designed and developed with classic technologies as well as innovative technologies that may be uncommon to the standard user. Combining all of them has been a long and fulfilling process, allowing us to learn new things. The technologies used on this project are listed below:

- Content Management System
  - HTML5, CSS3, Javascript for the frontend
  - PHP for the backend
  - MongoDB as the main database
- CMS Integration into Task Automation Server
  - Web Technologies: PHP, MongoDB
  - Semantic Technologies: Notation3, RDF, EYE, EWE Ontology
- Mobile App Improvement
  - Android

The software developed in this project has been based on, and integrated into two different projects from the Intelligent Systems Group:

- Development of a Task Automation Platform for Beacon enabled Smart Homes [15].
- Design and implementation of a Semantic Task Automation Rule Framework for Android Devices [7].

In the following sections we will describe in depth the achieved goals, the problems faced and some suggestions for a future work.

5.3 Achieved goals

In the following section we will go over the achieved goals during the making of this project:

- **Development of a Content Management System for Smart Offices.** This was the main objective in this project, designing and implementing a system which allows custom content creation for every user.
- **Design and implementation of a new Place ontology class.** In order to associate the content to different locations, we have defined a Place ontology class using the EWE Ontology.
- **Creation a graphical interface which allows users to create content in an easy intuitive way.** We have made sure that the process to create new content was really straightforward. This has been achieved through the creation of a visual builder using web technologies.
- **Integrating the CMS with the Task Automation Server codebase.** We have integrated our main project into the Task Automation Server in order to let the users access both features with the same login. This way users can manage their rules, channels and contents from a single place.
- **Creation of new Channels on the Task Automation Server.** New channels have been defined in order to achieve our goals. The events and actions of these channels are described in Notation3 and modeled after the EWE Ontology. The Place and CMS channel are now available on the TAS for everyone to use, improving the way rules are defined.
- **Definition of new EYE rules on the TAS.** New rules have been created for the automation of content delivery. These rules are written in Notation3, and they are evaluated by the EYE rule engine.

5.4 Problems faced

During the development of the project we had to face some problems. These problems are briefly described in this section:

- **Beacons accuracy:** The beacons owned by the Intelligent Systems Group are an early version of the commercial version of Estimote Beacons. The distance measures

between the mobile phone and the beacon deviated more often than it should. In order to overpass this obstacle, we tested all the beacons and used the more accurate ones for the development of this project. New versions of Estimote Beacons are reported to be more precise.

- **Task Automation Platform limitations:** EWE-Tasker main problem comes when we want to trigger multiple actions with a single event capture. Other task automations systems like Zapier are recently allowing to do this, so it is definitely a future line of work for EWE-Tasker. We have tackled this problem by linking two rules. The action triggered by the first rule will act as the event captured by the second one.
- **MongoDB PHP Controller and OSX compatibility**: the development of this project has been made on a Macbook Air with OSX El Capitan. The MongoDB controller for PHP is really tricky. This was solved by hosting the project on another UNIX server and accessing remotely to make changes.

5.5 Future work

Finally, in the last section we will talk about possible future lines of work that could be added to this project:

- **Collaborative Content Editing.** Giving the possibility to users of creating content together could achieve better results. It also would promote teamwork in the workplace.
- **Auto Save Function.** Many content editors over the Internet allow this (e.g. Google Drive). Making use of asynchronous HTTP requests every time major changes are made to the content, we could allow auto saving. This way users would not lose the progress made if they accidentally closed the editor.
- **Wearables adaptation.** Making use of Android Wear we could show the notifications in the smartwatch.
- **iOS application.** This could be achieved either by writing a native iOS application based on the Android one, or creating a hybrid app using React Native. Taking the hybrid approach there would be only one codebase to mantain.
- **Semantic Recommendation Framework.** Making even more precise through the use of a recommendation framework.

Bibliography

[1] Apache. Apache Stanbol, semantic content management. http://stanbol.apache.org/.

[2] Wojtek Borowicz. Bleesk uses beacons to promote local deals and grows exponentially. https://community.estimote.com/hc/en-us/articles/204412023-Bleesk-uses-beacons-to-promote-local-deals-and-grows-exponentially, 2015.

[3] M. Coronado. EWE Ontology Specification. http://www.gsi.dit.upm.es/ontologies/ewe/, 2013.

[4] International Data Corporation. Smartphone OS Market Share, 2016 Q3. http://www.idc.com/promo/smartphone-market-share/os;jsessionid=C0B86D70FEBB7A2936E47E7189480242, 2016.

[5] Ephox. TinyMCE, a javascript library for rich-text editing. https://www.tinymce.com/.

[6] Estimote. Estimote Beacons. http://estimote.com/.

[7] Antonio Fernández Llamas. Design and implementation of a semantic task automation rule framework for android devices.

[8] Jorge García Castaño. Future Students: ETSIT Open House Day, 2015.

[9] Pushmote Inc. Pushmote, mobile engagement. https://pushmote.com/.

[10] Zapier Inc. Zapier, automated actions for everyday tasks. https://zapier.com/.

[11] J.DeRoo. Euler, yet another proof engine. http://eulersharp.sourceforge.net.

[12] Rover Labs. Rover, location-powered mobile engagement. https://www.rover.io/, 2014.

[13] madebymany. SirTrevor, a rich content editor. http://madebymany.github.io/sir-trevor-js/.

[14] Inc. MongoDB. MongoDB, an open-source nosql database. https://www.mongodb.com/.

[15] Sergio Muñoz López. Development of a task automation platform for beacon enabled smart homes.

[16] Rackspace Studios, SFO. Y Combinator company, Estimote, shows why Low Energy Bluetooth is so important. https://www.youtube.com/watch?v=VfJch1XpCOw&feature=youtu.be&t=580, 2013.

[17] CKSource sp. CKEditor, a html/wysiwyg editor. http://ckeditor.com/.

[18] W3C Team. Notation3 (N3): a readable RDF syntax. https://www.w3.org/TeamSubmission/n3/, 2011.

APPENDIX A

Channel Creation

This appendix describes the new two channels created in the Task Automation Server as we stated in the section 3.4.1.

A.1 Place Channel definition

The channel Place is created with the following params:

- **Title:** Place.
- **Description:** This channel represents a specific place powered by beacons.
- **Nicename:** Place.
- **Event:**
  - Title: Inside Of.
  - Rule:

```
?event rdf:type ewe-place:InsideOf.
?event ewe:placeID ?placeID.
```

  - Prefix:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/#>.
@prefix ewe-place: <http://gsi.dit.upm.es/ontologies/ewe-place
    /ns/#> .
```

- **Action:**
  - Title: Entered.
  - Rule:

```
ewe-place:Place rdf:type ewe-place:Entered ;
ov:location "#PARAM_1#".
```

  - Prefix:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ewe-place: <http://gsi.dit.upm.es/ontologies/ewe-place
    /ns/#> .
@prefix ov: <http://vocab.org/open/#> .
```

A.2 CMS Channel definition

The channel CMS is created with the following params:

- **Title:** CMS.
- **Description:** This channel represents the custom Content Management System.
- **Nicename:** CMS.
- **Action:**
  - Title: Show.
  - Rule:

```
ewe-cms:CMS rdf:type ewe-cms:Show;
ov:message ?placeID.
```

  - Prefix:

```
@prefix ewe-cms: <http://gsi.dit.upm.es/ontologies/ewe-cms/ns
    /#> .
@prefix ov: <http://gsi.dit.upm.es/ontologies/ov/ns/#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```