

PROYECTO FIN DE CARRERA

Título: Development of a Common Sense Reasoning Agent for
Smart Home Automation
Autor: Alejandro López Fernández
Tutor: Carlos Ángel Iglesias Fernández
Departamento: Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente: Gregorio Fernández Fernández
Vocal: Mercedes Garijo Ayestarán
Secretario: Carlos Ángel Iglesias Fernández
Suplente: Marifeli Sedano Ruíz

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes



PROYECTO FIN DE CARRERA

**DEVELOPMENT OF A COMMON SENSE
REASONING AGENT FOR SMART HOME
AUTOMATION**

Alumno: D. Alejandro López Fernández

Tutor: D. Carlos Ángel Iglesias Fernández

2013

Resumen

Esta memoria presenta a 'Sensato', un sistema basado en conocimiento de sentido común cuyo objetivo es el de facilitar a los usuarios el desarrollo de tareas relacionadas con el hogar inteligente. Este sistema pretende ofrecer información acerca de cómo se usan los elementos empleados en las tareas del hogar inteligente, así como los pasos necesarios para completar dichas tareas, para su uso por personas o por otros elementos software. El sistema abarca todas las etapas del proceso, desde la adquisición del conocimiento, extracción de información contenida en él, la adaptación del mismo en una base de conocimiento, y la elaboración de un agente final. Profundizaremos en el procesamiento del conocimiento para cumplir con nuestros objetivos, así como en la obtención de métricas que permitan ver la eficiencia de nuestro sistema.

Para conseguir dichos objetivos, estudiaremos herramientas disponibles como Ollie, ReVerb o ConceptNet, entre otras, modificándolas para satisfacer nuestras necesidades. También desarrollaremos software propio para así lograr cumplir con la definición de objetivos propuesta.

Este documento muestra la arquitectura de la solución planteada, así como el detalle de la implementación de cada módulo. Además, se muestran varios posibles casos de uso de la plataforma, aunque su alcance no está limitado a estas.

Palabras clave: Agente inteligente, conocimiento de sentido común, hogar inteligente, base de conocimiento, extracción de información, adquisición de información, procesamiento de lenguaje natural.

Abstract

This thesis presents 'Sensato', a common sense knowledge system with the objective of helping the users in the development of Smart Home related tasks. This systems aims to offer information about how are some elements used in Smart Home Automation tasks, and give the necessary steps in order to complete that task, for the use of human beings or other pieces of software. The system covers all the stages of the process, starting from the knowledge acquisition, the extraction of the information contained on it, the adaptation int a knowledge base and the development of a final agent. We will discuss in detail the processing of the knowledge in order to fulfil our objectives, as well as the study of metrics that will help us to measure the efficiency of our system.

In order to achieve such objectives, we will study some tools such as Ollie, ReVerb or ConceptNet, among others, modifying them in order to satisfy our needs. We will also develop new pieces of software to accomplish the goals defined.

This document shows the architecture of the proposed solution, as well as the detail of the implementation of each module. Also, some example use cases of the platform are shown, although the scope is not limited to them.

Palabras clave: Intelligent Agent, Common Sense Knowledge, Smart Home, Knowledge Base, Information Extraction, Knowledge Acquisition, Natural Language Processing.

Glosary

In the next lines it is shown a list of the acronyms and abbreviations used within this project, with their respective meanings.

- **IE:** Information Extraction.
- **OIE:** Open Information Extraction.
- **KB:** Knowledge Base.
- **URI:** Uniform Resource Identifier.
- **API:** Application Programming Interface.
- **XML:** Extensible Markup Language.
- **JSON:** Javascript Object Notation.
- **NLP:** Natural Language Processing.

Agradecimientos

A mi familia, por haberme apoyado en todo el camino, aguantando mis peores días, y disfrutando con mis alegrías.

A todas las personas que integran el Grupo de Sistemas Inteligentes, por haberme ofrecido un ambiente inmejorable de trabajo. Vuestro apoyo y ayuda ha sido fundamental para la elaboración de este proyecto.

A mis amigos de la Universidad, porque habéis hecho que haya disfrutado cada día aquí. Me llevo recuerdos inolvidables, y buenos amigos para toda la vida.

A mis amigos del barrio, por hacer posible que desconectara durante los fines de semana, ofreciendo siempre risas y diversión. A veces no les pude dedicar todo el tiempo que se merecen, pero su amistad ha sido continua.

A mis amigos de Navalperal, por haber conseguido que haya pasado veranos increíbles.

A quien me hace sonreír cada día.

Contents

Resumen	v
Abstract	vii
Glossary	ix
Agradecimientos	xi
Índice de Tablas	xix
Índice de Figuras	xxi
1 Introduction	1
1.1 Motivation	2
1.2 Structure	2
2 Enabling technologies	5
2.1 Available General Purpose Knowledge Bases	6
2.1.1 DBPedia	6
2.1.2 WordNet	8
2.1.3 Freebase	8
2.2 Common Sense Knowledge Bases	10
2.2.1 ConceptNet	11
2.2.2 Common Sense Knowledge Acquisition	14
2.2.2.1 Verbosity	14

2.2.2.2	Common Consensus	16
2.2.2.3	20 Questions	16
2.2.2.4	Open Mind Common Sense	17
3	Architecture	19
3.1	Introduction	20
3.2	Knowledge acquisition	22
3.3	Knowledge extraction	22
3.4	Knowledge adaptation	24
3.5	Interaction Agent	24
4	Knowledge Acquisition for Sensato	27
4.1	Introduction	28
4.2	Game structure	28
4.3	Technologies used	29
4.3.1	PHP	29
4.3.2	Ajax	30
4.4	Gamification	30
4.5	Mixing the results	33
5	Information Extraction for Sensato	35
5.1	Background	36
5.1.1	TextRunner	36
5.1.2	Reverb	36
5.1.2.1	Syntactic Constraint	37
5.1.2.2	Lexical Constraint	38
5.1.2.3	Limitations	38
5.1.3	Ollie	38

5.2	Experiments	39
5.2.1	Reverb Results	41
5.2.1.1	Good extractions	42
5.2.1.2	Illogical extractions	42
5.2.1.3	No extractions	42
5.2.2	Ollie Results	43
5.2.2.1	Good extractions	44
5.2.2.2	Illogical extractions	44
5.2.2.3	No extractions	44
5.2.3	Reverb with unary option results	44
5.2.3.1	Good extractions	45
5.2.3.2	Illogical extractions	45
5.2.3.3	No extractions	46
5.3	Conclusions	46
5.4	Information Extraction for Sensato	47
6	Knowledge Adaptation for Sensato	51
6.1	Introduction	52
6.2	Knowledge base	52
6.2.1	Running the API Server	52
6.3	Inserting Smart Home Automation knowledge in ConceptNet	53
6.3.1	JSON	53
6.3.2	Edges	54
6.3.3	Knowledge as a list of steps	56
6.3.4	Knowledge as concepts	56
7	Interaction Agent for Sensato	59

7.1	Introduction	60
7.2	Use Cases	60
7.2.1	Use case 1	60
7.2.2	Use case 2	61
7.3	Implementation of the agent	61
7.3.1	Structure of the agent	61
7.3.1.1	KB Query	62
7.3.1.2	RESTful API	64
7.3.1.3	Web Agent	65
8	Conclusions and Future Work	69
8.1	Introduction	70
8.2	Conclusions	70
8.3	Future Work	71
8.3.1	Integrate the Knowledge Acquisition module in Social Networks . .	71
8.3.2	Acquire knowledge from websites	73
8.3.3	Better connection between different kind of knowledges	73
8.3.4	Integrating the agent with Jade/Jadex/Wade	73
8.3.5	Using NLP techniques in the agent	75
8.3.6	Integrating the web-agent with other systems	76
A	Tests sentences	77
A.1	Test sentences	79
B	Installing a local version of ConceptNet	81
B.1	Installation process	83
B.1.1	Setting up the enviroment	83
B.1.2	Installing Solr	83

B.1.3	Installing ConceptNet	84
B.2	Adding the knowledge from ConceptNet	84
C	Sorting algorithm	87
	Bibliografia	93

List of Tables

5.1	Examples of incoherent extractions	37
5.2	Uninformative relations (left) and their completions (right)	37

List of Figures

2.1	Overview of the DBpedia components	7
2.2	Semantic relations in WordNet	8
2.3	Example of the search 'car' in WordNet	9
2.4	Example of result of a query in Freebase	10
2.5	Some of the nodes and links in ConceptNet	11
2.6	ConceptNet 5	12
2.7	An quick example overview of some of the information about the concept "Towel" in ConceptNet 5	13
2.8	Structure of the hints in Verbosity	15
2.9	Development of the Common Consensus Game	17
2.10	20 questions learning from a concept	17
2.11	20 questions guessing a concept	18
3.1	Architecture of Sensato. The gears indicate new software developed within this project	21
3.2	Knowledge acquisition module	22
3.3	Knowledge extraction module	23
3.4	Knowledge adaptation module	24
3.5	Interaction Agent module	25
4.1	Platform where the users could contribute with their knowledge about Smart Home Automation	29
4.2	The welcome page to the game, asking for the user's name	31

4.3	The score of the game, shown to the user	32
4.4	Badgets on the game	32
5.1	Summary of the results given by Reverb	41
5.2	Summary of the results given by Ollie	43
5.3	Summary of the results given by Reverb using the unary option	45
5.4	Summary of the results of the three systems	46
5.5	Summary of the results of the new system	48
5.6	Comparison of all the systems analysed	48
5.7	Summary of the precision, recall and F-score of all the system analysed . .	49
7.1	First use case of the system.	60
7.2	Second use case of the system.	61
7.3	Structure of the agent	62
7.4	Running the Python module	63
7.5	Agent answering what HDMI is.	66
7.6	Agent answering what HDMI is used for.	67
7.7	Agent helping the user to perform a Home Automation Task	68
8.1	Penetration of social platforms.	72
8.2	Steps about a Smart Home Automation task by wikiHow.	74
8.3	JADE Agents platform.	75
8.4	Arquitecture of personal agent solution.	76
B.1	ConceptNet running on a local server	86

CHAPTER 1

Introduction

1.1 Motivation

We live in a connected world. New technologies have arrived in our life with great intensity, and most of us can not imagine a world without Internet, smartphones or digital cameras.

These technologies allow us to perform some tasks that, years ago, were unbelievable: we can buy tickets for a concert from our computer, or watch the latest episode of our favourite TV show from our cell phone. These technologies can also be applied into tasks within our home, to perform what is called Smart Home Automation tasks.

Home Automation is defined as the automation of the home, housework or household activity. A home automation system integrates electronic devices in a house with each other. Devices may be connected through a computer network to allow control by a personal computer, and may allow remote access from the internet. Through the integration of information technologies with the home environment, systems and appliances are able to communicate in an integrated manner which results in convenience, energy efficiency, and safety benefits.

For example, we may want to watch a film in the TV. We will have to download the film, copy it to our tablet, get a HDMI cable, connect the tablet to the TV... That are a lot of steps, with a lot of concepts we may be (or not) familiar with.

The scope of this project is to develop a system capable of acquiring knowledge about Smart Home, identifying the steps needed in order to perform a Smart Home Automation process, and help the user when any doubt arises.

There are many possible actors that can take advantage of our system. For example, another software could read the tasks in order to achieve some goal, and perform all the steps that it can do. Another example is a person that wants do perform some task, but does not know how to do it. Our system would show him all the steps needed, and will provide further information.

1.2 Structure

This document is divided into eight chapters. In the following lines we will summarize their content, to make the reading of this document easier.

- **Chapter 1** presents the problem that we aim to solve, and describes the structure of this document.

- **Chapter 2** shows the previous work that has been done about the topic related to the project. It also describes some of the technologies used within this project.
- **Chapter 3** describes the software architecture of the solution, and its component parts. It also gives a little description about the modules that form the final solution.
- **Chapter 4** talks about the method we have used to acquire Smart Home Automation knowledge.
- **Chapter 5** presents the techniques used in order to extract relations and knowledge within the information retrieved by the previous module.
- **Chapter 6** shows how we have inserted the knowledge from previous steps into our knowledge base.
- **Chapter 7** describes the agent that can interact with the knowledge base, in order to present it to the final user.
- **Chapter 8** discusses about the conclusions of the project, and describes futures lines of work related with this project.

At the end of the document, we have included some Appendix in order to clarify some of the aspects of this project.

- **Appendix A** shows some of the corpus used in the development of this thesis for testing. It can help us to clarify the domain of the solution provided, and it allows us to reproduce the results.
- **Appendix B** explains in detail the necessary steps in order to deploy a local version of ConceptNet. As we will see, this process is not documented well enough, so this appendix can help other people with the same problem.
- **Appendix C** is dedicated to an algorithm used for sorting some elements within an array. The code is shown in case someone wants to test it, and it is commented in order to make it legible for other people.

CHAPTER 2

Enabling technologies

In the development of this project several different technologies were used and analysed. In this chapter we explain some of these technologies, in order to get a clear view of the existing knowledge on this topic.

2.1 Available General Purpose Knowledge Bases

General purpose knowledge bases are a set of databases that provide information that a group of people has gathered about transversal knowledge, such as definitions or general facts.

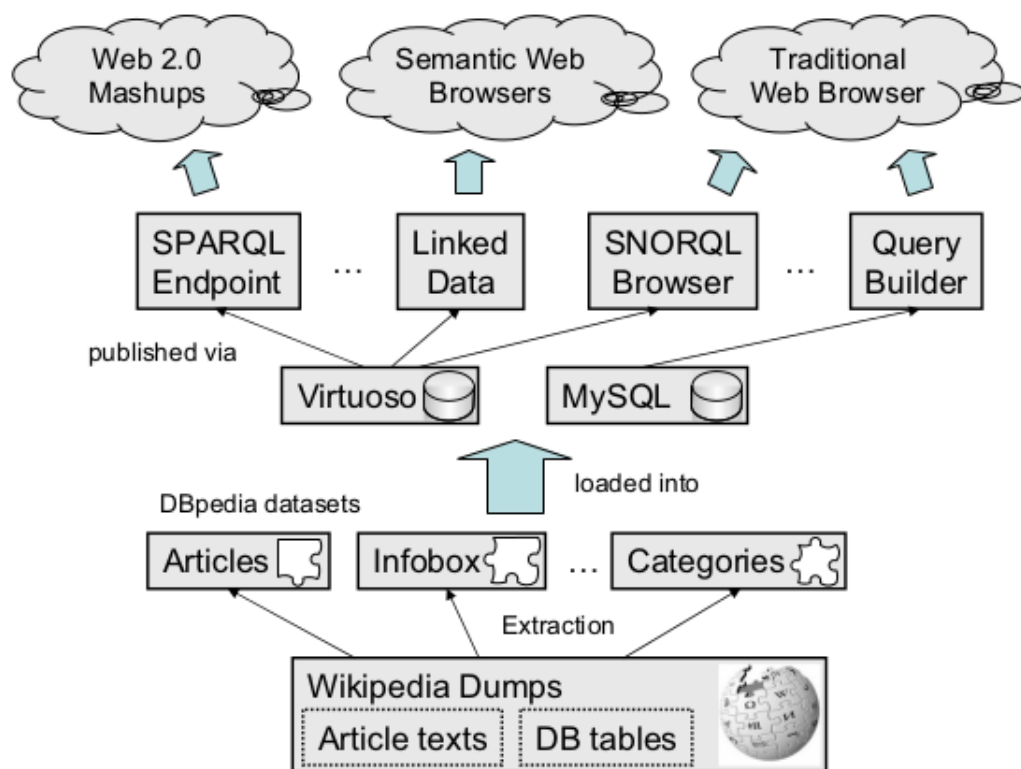
2.1.1 DBPedia

DBpedia [1] is a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web. As of September 2011, the DBpedia dataset describes more than 3.64 million things, out of which 1.83 million are classified in a consistent ontology, including 416,000 persons, 526,000 places, 106,000 music albums, 60,000 films, 17,500 video games, 169,000 organizations, 183,000 species and 5,400 diseases. The architecture of DBPedia can be seen in Fig. 2.1

The knowledge available in DBPedia is obtained parsing Wikipedia article texts with a PHP Software. When it has been added to DBPedia, it can be accessed in three different forms.

- **Linked data:** Linked Data is a method of publishing RDF data on the Web that relies on `http://` URIs as resource identifiers and the HTTP protocol to retrieve resource descriptions. The URIs are configured to return meaningful information about the resource (typically, an RDF description containing everything that is known about it).
- **SPARQL:** SPARQL is a query language for databases, able to retrieve and manipulate data stored in RDF. Client applications can send queries over the SPARQL protocol to an endpoint at `http://dbpedia.org/sparql`.
- **RDF Dumps:** N-Triple serializations of the datasets are available for download at the DBpedia website and can be used by sites that are interested in larger parts of the dataset.

Figure 2.1: Overview of the DBpedia components



2.1.2 WordNet

WordNet [2] is a lexical database created at the Cognitive Science Laboratory of Princeton University.

WordNet creates unordered sets of synonyms called '*synsets*'. These synsets are inter-linked by means of conceptual-semantic and lexical relations

WordNet provides different kinds of semantic relations between these synsets. These relations can be seen in Fig. 2.2

Figure 2.2: Semantic relations in WordNet

Semantic Relation	Syntactic Category	Examples
Synonymy (similar)	N, V, Aj, Av	pipe, tube rise, ascend sad, unhappy rapidly, speedily
Antonymy (opposite)	Aj, Av, (N, V)	wet, dry powerful, powerless friendly, unfriendly rapidly, slowly
Hyponymy (subordinate)	N	sugar maple, maple maple, tree tree, plant
Meronymy (part)	N	brim, hat gin, martini ship, fleet
Troponymy (manner)	V	march, walk whisper, speak
Entailment	V	drive, ride divorce, marry
<i>Note:</i> N = Nouns Aj = Adjectives V = Verbs Av = Adverbs		

WordNet can be accessed via its website or downloaded into a computer. In Fig. 2.3 we can see a search example performed in the website. The term searched is 'car', and each row shows a different synset of this word. Semantic and lexical relations can be displayed by clicking on the links.

2.1.3 Freebase

Freebase [3] is an open, Creative Commons licensed repository of structured data of almost 23 million entities, which are a single person, place, or thing. Freebase connects entities together as a graph.

It contains data extracted from sources such as Wikipedia, ChefMoz, NNDB, and MusicBrainz, as well as individually contributed data from its users.

Figure 2.3: Example of the search 'car' in WordNet

WordNet Search - 3.1

[WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- **S: (n) car**, [auto](#), [automobile](#), [machine](#), [motorcar](#) (a motor vehicle with four wheels; usually propelled by an internal combustion engine) "*he needs a car to get to work*"
- **S: (n) car**, [railcar](#), [railway car](#), [railroad car](#) (a wheeled vehicle adapted to the rails of railroad) "*three cars had jumped the rails*"
- **S: (n) car**, [gondola](#) (the compartment that is suspended from an airship and that carries personnel and the cargo and the power plant)
- **S: (n) car**, [elevator car](#) (where passengers ride up and down) "*the car was on the top floor*"
- **S: (n) cable car**, **car** (a conveyance for passengers or freight on a cable railway) "*they took a cable car to the top of the mountain*"

Freebase provides an interface to connect external software to its database. This way another application can perform search and queries against Freebase's data or add new data to the graph, helping to make it bigger.

For example, in Fig. 2.4 we can see a query in which we asked for a list of car companies and the number of employees of each one of these companies.

Figure 2.4: Example of result of a query in Freebase

Name	number
BMW	106,179
Chrysler Group LLC	58,000
	132,130
Ford Motor Company	245,000
Honda Motor Company, Ltd	144,785
	178,960
Nissan Motor Co., Ltd.	163,099
	206,484
Toyota Motor Corporation	299,394
	316,121
AB Volvo	101,698
Volkswagen Group	369,928
Daimler AG	272,382
Fiat	185,227
British Aerospace	127,000
	67,000
Renault	129,068
PSA Peugeot Citroën	207,800
	201,690
Delphi Automotive	169,500
	171,400
Denso	119,000

2.2 Common Sense Knowledge Bases

'A lemon is yellow.' *'You must open the door to get to another room.'* *'The TV should be on in order to watch the news'*

These sentences can explain a kind of knowledge: Common Sense Knowledge. It contains a lot of facts or rules that are simple, evident and that every *person* knows. But computers do not have this kind of knowledge that, as we can see, is so important in order to make good decisions. Otherwise, a computer software could, for example, suggest a user to go from America to Spain by car. We all know that nobody can perform such thing, because

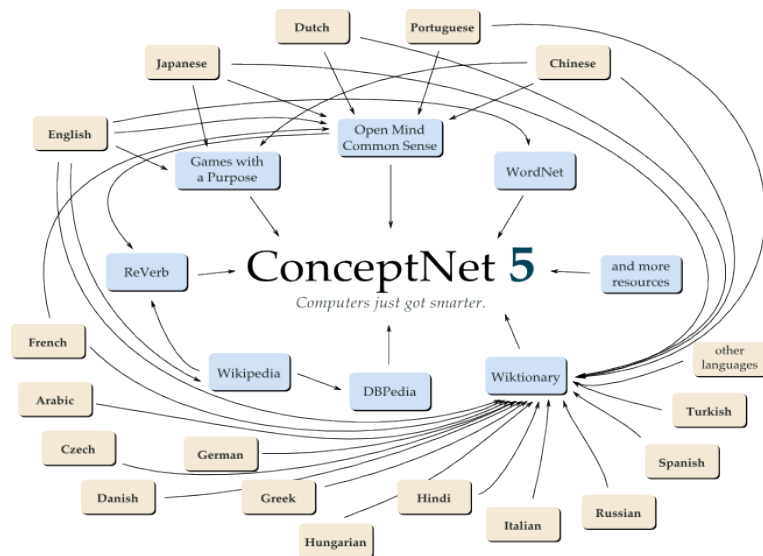


Figure 2.6: ConceptNet 5

project at the MIT Media Lab. Divisi, as any other reasoning method applied on ConceptNet, can infer additional relations that are not stated in the network, but that are also common sense knowledge.

At the time of this writing, ConceptNet is already in its fifth version. Several changes have been made to the system, as well as several new sources of data have been included in the data acquisition procedures. Thus, the knowledge base has not only grown beyond the limits of the former ConceptNet 4 database, but has also been significantly enriched through other changes.

Originally, ConceptNet's semantic network was a graph in which nodes represented concepts and links (or *edges*) gave information on their relation. While this still applies to ConceptNet 5, the network has grown to become a *hypergraph*. In other words, edges can now act as nodes and have other edges pointing to them. We can have *edges about edges* which not only provides a much larger network, but a much richer one. Thus, relations will have additional information from which we can infer its accuracy, reliability or justification.

Furthermore, relations are no longer limited to the specified set we had in previous versions. While it keeps being advisable to stick up to normalised sets when possible, additional relations can be created at convenience. In this direction, the fact of treating relations as nodes themselves comes in handy as a way of counterfeiting increased data sparsity or analogous relations which could indeed become a drawback on the quality of the network.

While previous versions of ConceptNet expressed nodes and relations exclusively using

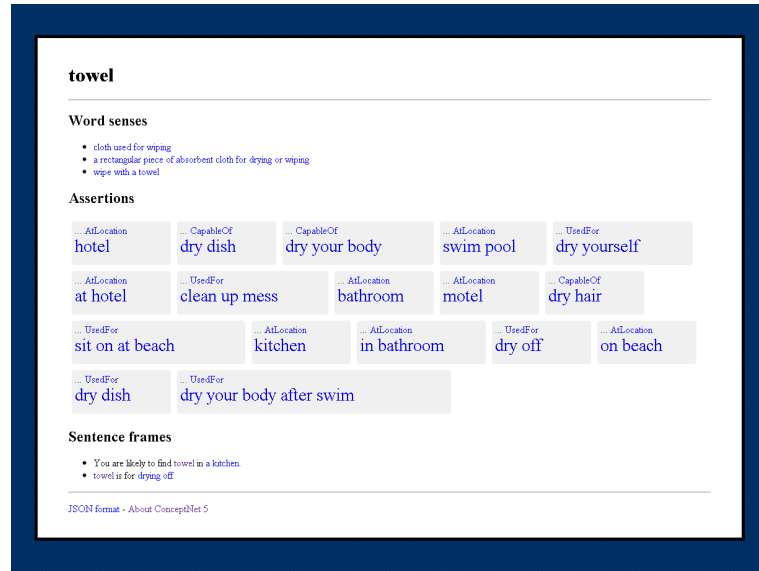


Figure 2.7: An quick example overview of some of the information about the concept “Towel” in ConceptNet 5

English language, ConceptNet 5 aims to represent also interlingual knowledge. For this purpose, a new type of relation “TranslationOf” has been added to the set of basic relations. This way, all the relations existing between English terms are automatically reachable from their translations through their “TranslationOf” edges, being very handy for non-English or multilingual applications. Furthermore, the translated nodes can be very useful themselves, in order to find the best translation of a word in a given context. As we mentioned before, any node can be an edge in ConceptNet 5. Having different language nodes could become a problem when being used to state relations. However, basic abstract notions such as “MadeOf” are still used in their English form to mean the same across all languages, and other language-specific edges can still be used in an interlingual context if the appropriate “TranslationOf” edges exist.

Furthermore, the acquisition procedures for the common sense statements have also significantly changed since the first versions of ConceptNet. While initially relying uniquely on the Open Mind platform, and in users actively *teaching* things to the system, ConceptNet 5 also gathers knowledge from several RDF sources such as DBPedia (Sec. 2.1.1) and even from general purpose websites using ReVerb [5] to extract relational knowledge from wikipedia and other sources.

2.2.2 Common Sense Knowledge Acquisition

Common Sense Knowledge acquisition is the process of gathering this kind of knowledge from people, and representing it in a structured form that a computer can use.

This is a hard process, for two main reasons:

- The amount of common sense knowledge is huge, so is difficult for a group of people to extract it all. It is estimated that it would take 350 man-years of effort.
- A single piece of common sense knowledge can be expressed in a lot of different ways. It is needed to structure all this information and represent it in one standard way.

There has been some efforts on gathering knowledge by a group of people, such as CyC [6]. Nevertheless, as we said earlier, the amount of work for these people is huge, so the main strategy used nowadays in order to collect knowledge is by using gamification.

As said on Wikipedia, *gamification is the use of game thinking and game mechanics in a non-game context in order to engage users and solve problems*. Therefore, there has been some attempts to create games on which the users have fun while common sense knowledge is collected. Nevertheless, there are some problems with this approach:

- The game developed may not be entertaining enough in order to attract many users.
- The game may be not well spread, so only a few people know about it and the answers are not diverse enough.
- The answers the users provide may contain spelling mistakes.

In the next sections, a few of these games are analysed, in order to get an idea on the approach of this method.

2.2.2.1 Verbosity

Verbosity [7] is an online game for two persons. The first one is the narrator, and the second one is the guesser. The narrator is given a word, which the guesser can not see, and his duty is to give hints to the guesser in order for him to guess it.

The hints follow a pattern which the narrator must follow. This way, parsing the knowledge when the game has finished is easier. The structure of the hints can be seen in

Figure 2.8: Structure of the hints in Verbosity

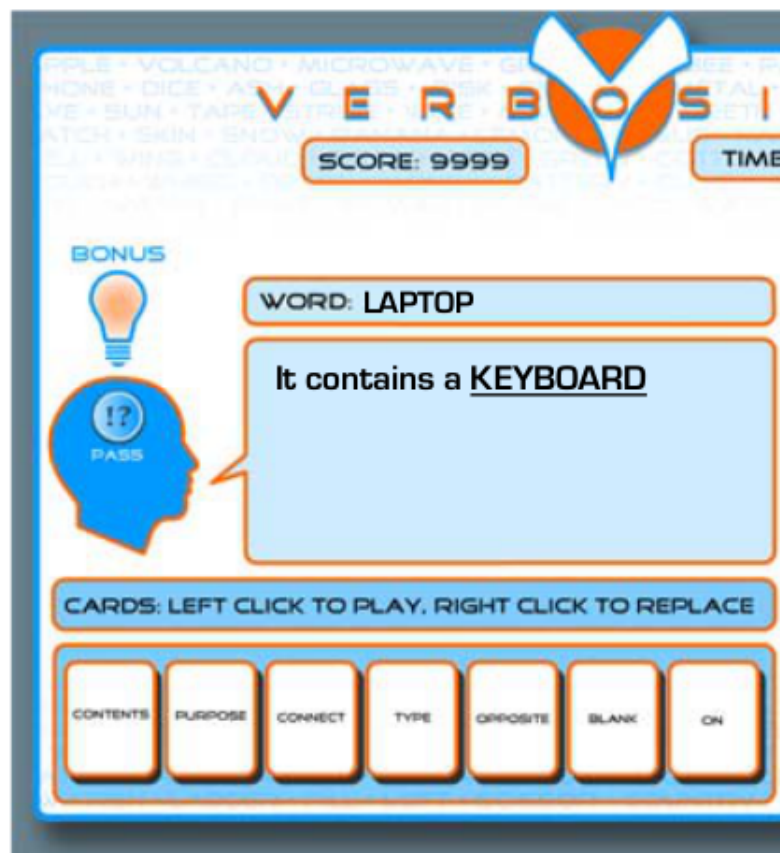


Fig. 2.8. Every time a hint has been given, the other player can try to guess the concept. If the answer is not the correct one, the narrator can give feedback, stating if the answer is *hot* or *cold*.

The game lasts until the concept has been correctly guessed, or until the time is up, in 6 minutes. Then, the guesser and the narrator change their roles.

When the players get tired, the game finishes. Common sense data can be collected using the hints given by the narrator. The main advantage is that, given that the hints follow a known pattern, the parsing is quite easy.

2.2.2.2 Common Consensus

Common Consensus [8] is a knowledge acquisition game developed by the MIT Media Lab. During this game, users receive questions such *Why would you want to X?* or *About how long would it take to X?*. In questions such as the later, only a few possible answers are available to the user. Common Consensus bases its reliability in the number of consenting answers. Therefore, the score of the game is obtained by multiplying the number of unique users who provided the same answer.

The aim of this game is to obtain a list of things that shape a plan. The type of questions of this game are oriented to get a list of goals and sub-goals in order to accomplish any kind of job.

When a question has been answered a sufficient number of times, it is observed that there are a certain number of goals and sub-goals more common than others. This data can be retrieved in order to enlarge our knowledge base.

2.2.2.3 20 Questions

20 questions [9] is a knowledge acquisition game developed for increasing the data available in the Open Mind Common Sense, Sec. 2.2.2.4.

20 questions provides two kinds of game. In the first one the game displays a series of generic questions about a known concept, and the user answers this questions with *Yes, No, Maybe, or Doesn't make sense*. That way the system can infer knowledge with the answers of the users, improving the previous knowledge available of the concept. An example of this can be seen in Fig. 2.10. In the second one, the users thinks mentally of a concept, and the systems ask questions. The system attempts to guess the user's concept after collecting 12, 15, and 18 pieces of information. It guesses once after the 12th and 15th

Figure 2.9: Development of the Common Consensus Game

Off the top of your head,
enter as many answers in as you can:

What are some things you would use to:
watch a movie?

12 seconds left

sound sy

movie

dvd ☒

television ☒

tv ☒

[Bad Question?](#)

pieces of information, and 3 times after the 18th. The system always guesses the current most likely concept. If the system has not guessed the concept after 20 questions, the user is asked to enter the concept and knowledge about this concept is automatically inferred based on the previous questions and answers. An example of this type of game can be seen in Fig. 2.11

Figure 2.10: 20 questions learning from a concept

Current knowledge

→ [Paprika](#) is a [spice](#). by  [dashro](#) Score: 2  

Page 1 of 1 (1 total)

Tell me more about paprika...

Is it an example of place? **No**

Are you likely to find it in a store? **Yes**

Are you likely to find it in a desk? **No**

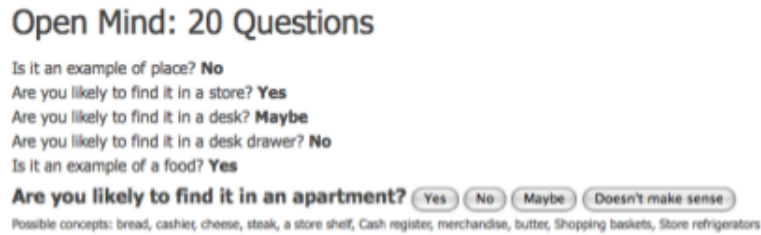
Are you likely to find it in dinner? **Yes**

Is it an example of a food?

2.2.2.4 Open Mind Common Sense

Open Mind Common Sense [10] is a knowledge acquisition system designed to acquire common sense knowledge from the general public over the web. Assuming that common sense is something that virtually everybody around the world has, it is of essential importance to approach the largest possible amount of human sources to acquire this knowledge.

Figure 2.11: 20 questions guessing a concept



The system counts on a basic web interface through which public users can submit pieces of knowledge. In order to target the biggest possible group, a system must be made so that it does not require formal training in computer science. Other systems such as Cyc [6] relied on formal languages, thus limiting the amount of input sources to people actually knowing the language and terms in the ontology. The Open Mind Common Sense system allows users to submit their assertions in free-form natural language, which is later processed through diverse techniques to convert English assertions into standard knowledge representations.

CHAPTER 3

Architecture

3.1 Introduction

The aim of this project is to develop a complete system capable of helping the users on performing Smart Home tasks thanks to the Common Sense Reasoning. The system is divided on two subsystems:

- **Frontend:** Subsystem responsible for the interaction with the users.
- **Backend:** Subsystem responsible for the data processing.

Each subsystem contains two modules. This modular division allows us to make changes on some parts of the project, without requiring a full change of the system. For example, we could add another knowledge acquisition module and, as long as it respects the interfaces used, it will automatically work on the project. A complete diagram of the project can be seen in Fig. 3.1.

In the following sections we will discuss further the objective of each of the modules seen on that figure.

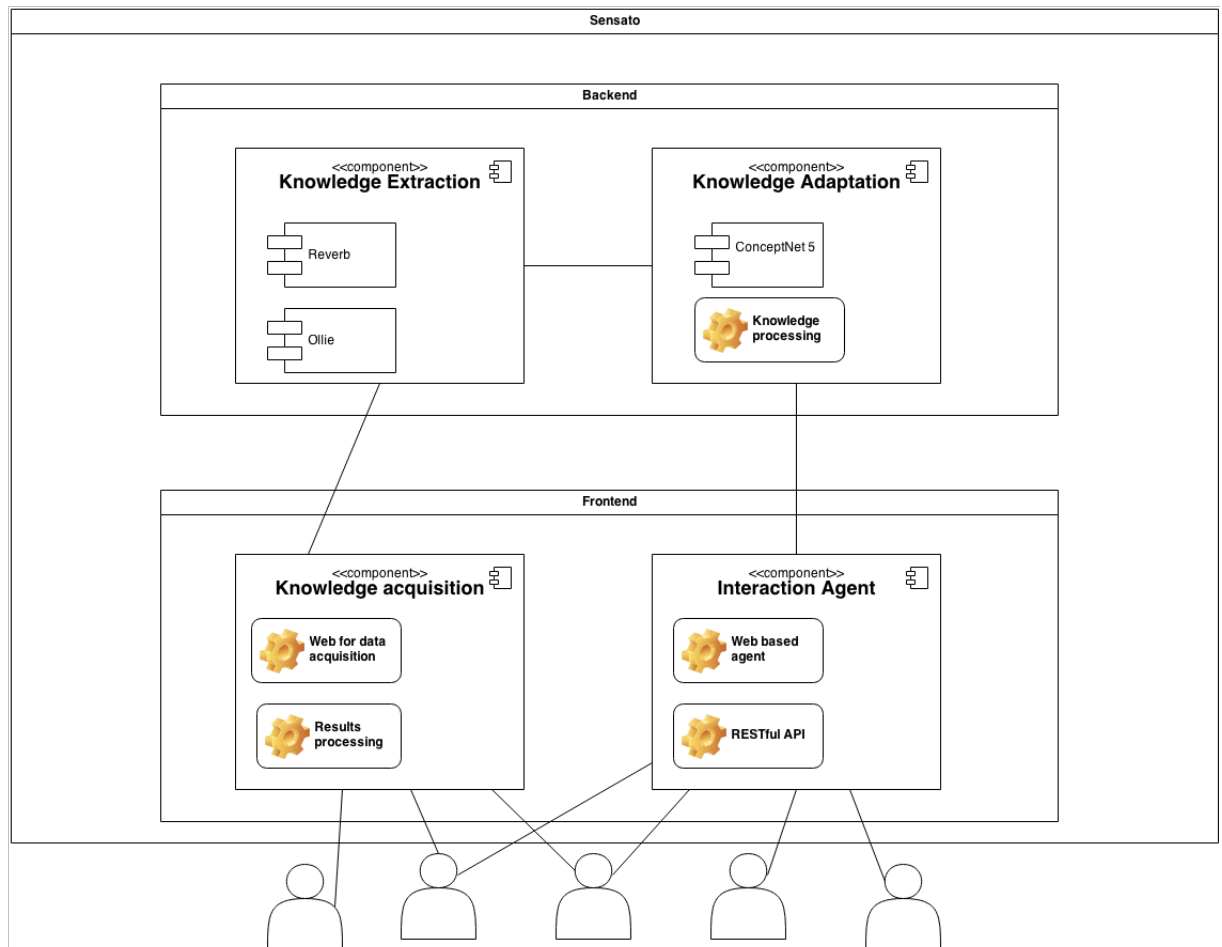


Figure 3.1: Architecture of Sensato. The gears indicate new software developed within this project

3.2 Knowledge acquisition

In this stage, we will collect all the necessary information needed in order to create a powerful database of knowledge about Smart Home Automation.

Although there are some websites where this information could be extracted [11, 12], this data is sometimes unspecific or wrong. Therefore, we have chosen to use a crowd-sourced game to collect the information, following the example of the games seen in Sec. 2.2.2. If the game is used by a sufficient number of users, the information obtained will be more accurate than the extracted from websites. Another advantage is that we could adapt the game in order to obtain knowledge about some specific topic, eliminating the need of waiting until these websites add that information.

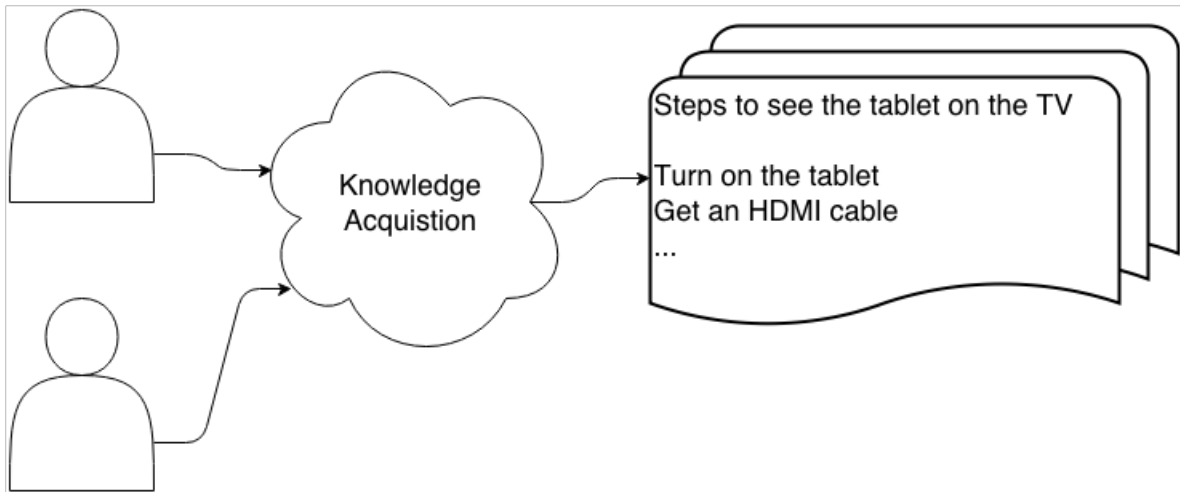


Figure 3.2: Knowledge acquisition module

This module will be discussed in Sec. 4

3.3 Knowledge extraction

This module is responsible for extracting all the relationships in the information extracted by the previous module. By performing this process, the system can learn about the elements used in Smart Home Automation processes, about their relationships and possible uses.

In this module we have chosen to integrate previous solutions, as the development of new tools could be expensive and inefficient.

This stage is explained in Sec. 5.

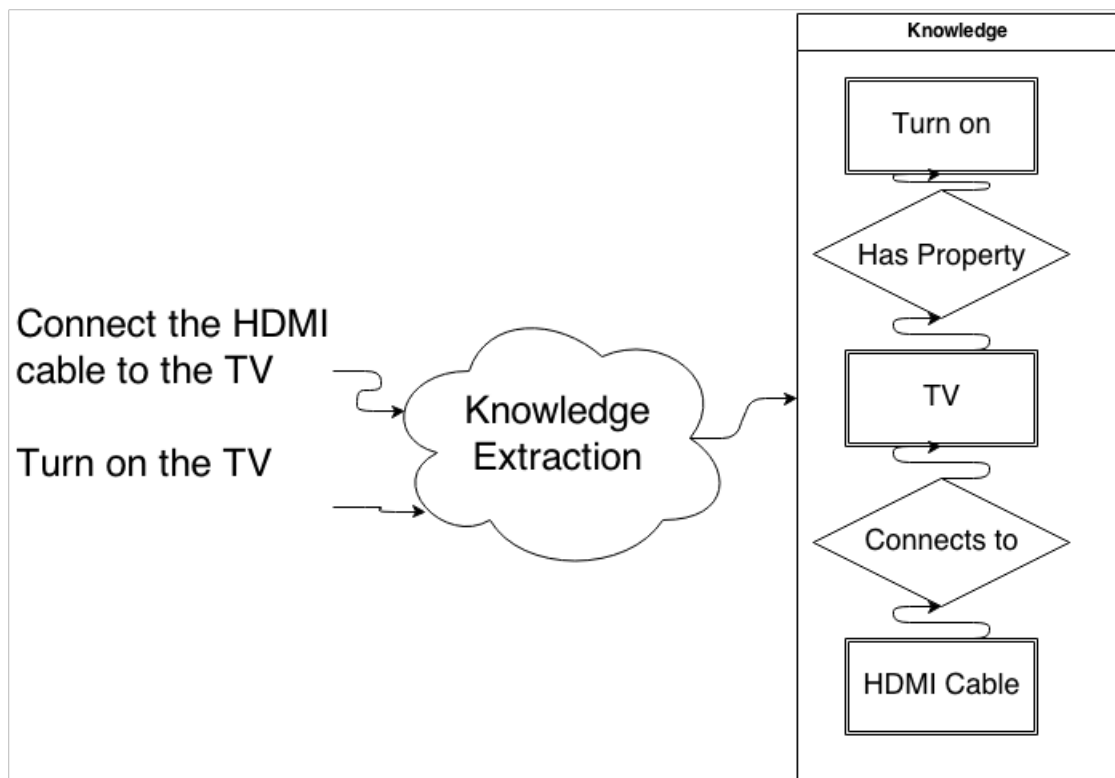


Figure 3.3: Knowledge extraction module

3.4 Knowledge adaptation

Once we have retrieved the steps needed to perform a task and the knowledge contained on it, we should add this information to a knowledge base.

This module is responsible for the adaptation of the knowledge into a standardized format, and its addition into a knowledge base.

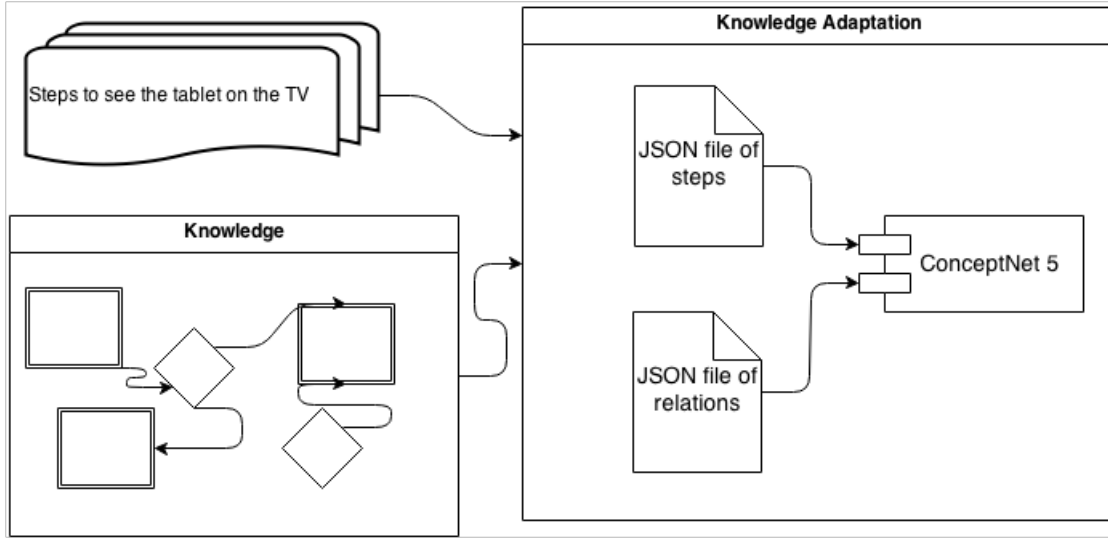


Figure 3.4: Knowledge adaptation module

With this module, we can make the chosen knowledge base *wiser*, and we can also benefit from the knowledge that has already been added into this knowledge base.

This module is explained in Sec. 6.

3.5 Interaction Agent

This module is responsible for the interaction with the final user.

In this stage, we develop an agent that connects with the knowledge base of the previous module. The user can ask this agent about the steps to perform some task, or inquire about further information about some element.

This module is explained in Sec. 7.

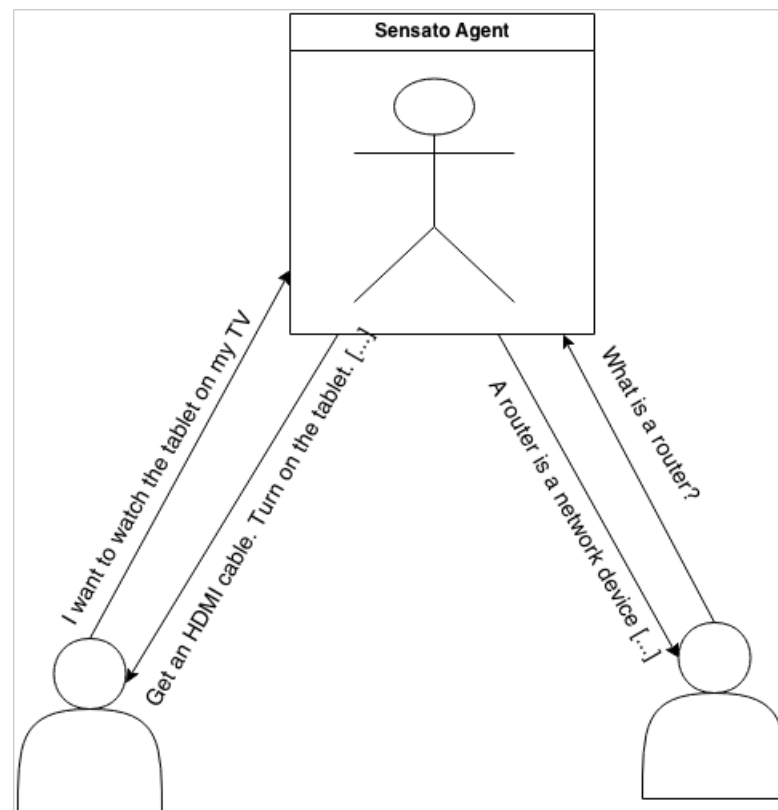


Figure 3.5: Interaction Agent module

Knowledge Acquisition for Sensato

4.1 Introduction

As a part of the project, we had to retrieve information about Smart Home Automation, specifically which are the steps needed to perform a task.

The knowledge about Smart Home Automation is vast and it could not be written by a only person, as said in Sec. 2.2.2. The approach taken by this project was to retrieve the data from multiple people, by letting them participate on a game which will help us to learn about this topic. The approach is based on gamification.

4.2 Game structure

In order to let the people help us, we developed a form where the users could insert the steps that they would make in order to complete some Home Automation related task.

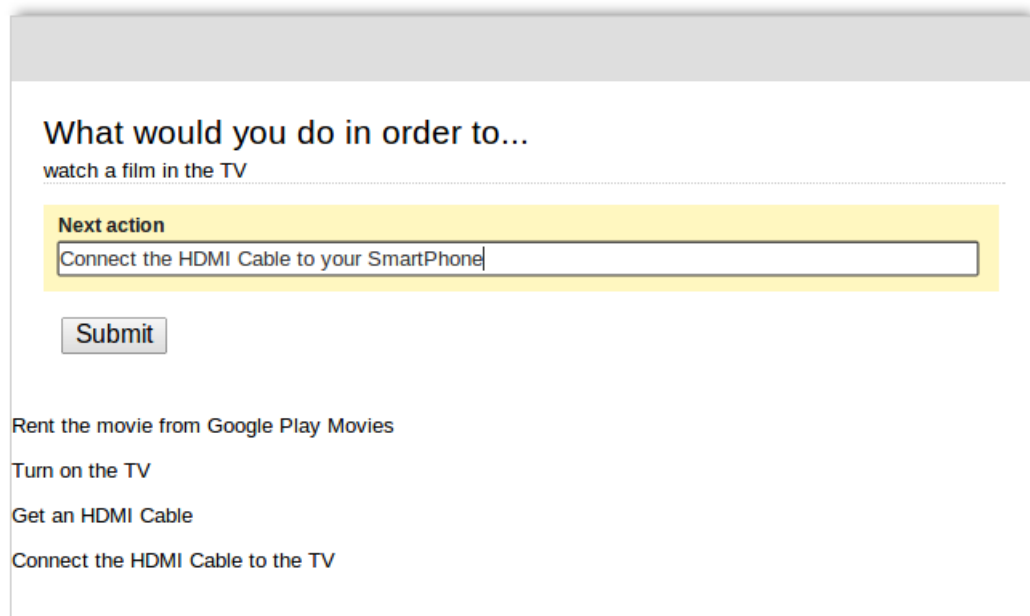
On this platform, the user is given a question, and is asked to give a complete list of steps he would perform in order to complete that task. As he submits new tasks, they appear on the screen and are recorded in a file for further processing.

Once the user thinks he has given all the necessary steps, he could continue with other questions or finish the process.

The final platform can be seen in Fig. 4.1.

As we have seen, the game asks the user questions related to Home Automation processes. These questions are obtained from a text file, and the system chooses randomly which question to ask. This approach has the advantage that, if necessary, new questions could be easily added to the system just modifying that text file.

This file, during the development of this project, had questions such as *How would you watch a film in the TV?* or *How would you connect the tablet to the PC?*



What would you do in order to...

watch a film in the TV

Next action

Connect the HDMI Cable to your SmartPhone

Submit

Rent the movie from Google Play Movies

Turn on the TV

Get an HDMI Cable

Connect the HDMI Cable to the TV

Figure 4.1: Platform where the users could contribute with their knowledge about Smart Home Automation

4.3 Technologies used

For the development of the mentioned platform different technologies were used.

4.3.1 PHP

PHP [13] is a server-side scripting language designed for web development but also used as a general-purpose programming language. PHP is now installed on more than 244 million websites and 2.1 million web servers. While PHP originally stood for Personal Home Page, it now stands for PHP: Hypertext Preprocessor.

PHP code is interpreted by a web server with a PHP processor module, which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data.

We chose PHP as the programming language because of its ease of deploying it on almost any web server. For the purposes of this project, our PHP files were deployed on a personal computer with a LAMP [14] (Linux, Apache, mySQL and PHP) solution installed.

4.3.2 Ajax

Ajax [15] stands for Asynchronous JavaScript and XML. With Ajax, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behaviour of the existing page. Data can be retrieved using the XMLHttpRequest object.

We used Ajax (in combination with jQuery) with the purpose of automatically update the platform when the user inserts a new step for a given task, so that he can see what he has already suggested.

4.4 Gamification

As we told in Sec. 2.2.2 , the use of a gamification process can boost the number of people helping us on getting knowledge. What is more, this number can grow exponentially when mixed with integration with social networks.

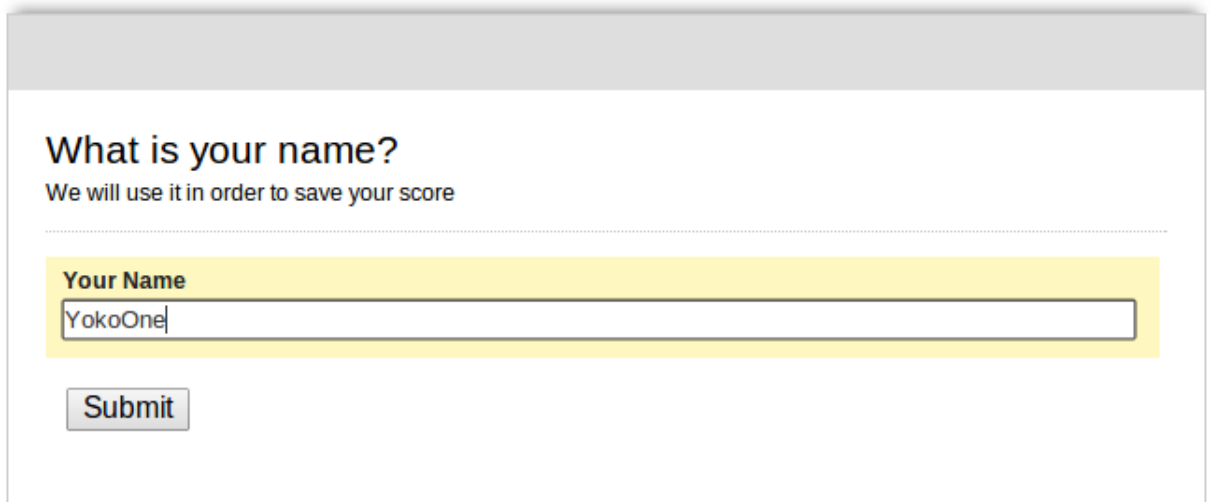
According to some studies [16], the contribution of users on *gamified* projects rises 15%, compared to normal projects. Also, the number of monthly visits to the site can double, as does the average number of minutes on the application. With all that information in mind, it seems like a good idea to 'gamify' our platform.

The first step was creating an environment which could identify the user. We needed that information in order to store the scores, so the individuals could get some rewards when they achieve a determined score. Therefore, the new welcome page is as seen in Fig.4.2:

The purpose of our game is to get the maximum number of steps in order to achieve some Home Automation task. The approach taken in our project was to assign a number of points to the participants according to the following rules:

- For each step added by an user, he would get 10 points. This way we promote that users create a lot of steps, so we can obtain a lot of data.
- For each steps added, which has also been added by other users, he would get only 5 points. This way, the users are asked to think creatively , so we can obtain knowledge that is not available in other forms.

Nevertheless, when testing the platform with real users, we realized that there might be a problem with this approach. The users are asked to input as many steps as they like, and



The image shows a web form with a light gray header bar. Below the header, the text 'What is your name?' is displayed in a bold, black font. Underneath this, in a smaller, regular black font, is the text 'We will use it in order to save your score'. A horizontal dotted line separates this text from the input area. The input area has a yellow background and contains the label 'Your Name' in bold. Below the label is a text input field with the text 'YokoOne' entered. At the bottom of the yellow area is a gray button with the text 'Submit' in bold.

Figure 4.2: The welcome page to the game, asking for the user's name

they always get points when adding new steps. But, as we can not automatically detect if this new steps are 'logical', the users could be inserting a lot of nonsense information.

Therefore, as a way to prevent that users added information that was not related to the question we were asking, the following rule was added;

- At least a 10 % of the steps added by an user should have been already added by other users. Otherwise, he would get a 0 points score.

Finally, the user can see his score in the same page, in order to know how he is performing, as seen in Fig. 4.3.

As a way to motivate the users when they achieve a high score, we show some 'badgets' to those users , as seen in Fig. 4.4.

Your score is 10.

Hello YokoOne. What would you do in order to...


connect the tablet to the PC

Next action

Submit

Figure 4.3: The score of the game, shown to the user

Your score is 1045.



Hello Alex. What would you do in order to...

connect the tablet to the PC

Next action

Submit

Figure 4.4: Badgets on the game

4.5 Mixing the results

Once that we have retrieved enough information (the amount of time needed may vary depending on the effectiveness of the game), we should have a list of plain text files, one for each question and person who answered that question.

The aim of this step of the project is to merge all the answers to each question in only one file per question. This way we will be able to process that information in the following stages of this project.

In the development of this task, we observed two major constraints:

- There may be some answers to some of the questions that are not *popular* enough. That is, some of the users may have introduced some steps to perform a task, but the rest of the people don't seem to agree with him. This can be caused because they are not really steps needed to complete the task, or maybe because these answers are provided by people more innovative than the others.
- Each user may have inserted the steps in a different order. As we are treating with a list of steps, the order seems important, and therefore some solution had to be provided.

To solve the first constraint, it seemed like a good idea to create a *threshold*, a minimum percentage of files containing each step in order to be added into the file containing the steps provided by all the users. We developed a Java program in order to do so, where the threshold can be configured. This way, we could make a system which only accepts steps taken by consensus (threshold near 100%) or a system with a lot of diversity (threshold near 0 %).

For the purposes of this project, we thought that the threshold should be around 20%. The main counterpart is that, by eliminating the steps that were added by less than 20% of the total, we could be eliminating some of the more innovative tasks, or some of the tasks that are so *obvious* to people that not many of them added them.

When running this program, each file was filtered, removing those sentences which were not repeated at least the threshold percentage.

The second problem also required some actions in order to solve it. We developed a Java program that executed an algorithm to join all the different files.

At first, we only have one file per user and question, and the 'uncommon' steps have

been removed, as stated before. At the end of the execution of the program, we should have only one file per question, with all the steps merged in correct order.

The algorithm can be shown in Sec. C.

Information Extraction for Sensato

5.1 Background

Once that we have obtained a trustful list of tasks needed to perform a Smart Home Automation Task, the next stage in our project is to retrieve the knowledge and relations present in those sentences.

Information Extraction (IE) is the task of extracting assertions from massive corpora. At first, the Information Extraction systems were based based on training on specific datasets, which was not scalable and hardly reliable. Those systems learned an extractor from labelled training examples. Modern systems use Open Information Extraction [17], which identify *relation phrases* and are capable of extracting arbitrary relationships on large amount of data. This approach removes the restriction of needing a pre-specified vocabulary. Relation phrases are usually a tuple of the form $t = (e_i, r_j, e_k)$ where e_i, e_k are entities and r_j is the relation between them.

In the development of our system, a number of Open IE software was analysed, with the aim of measuring their efficiency and testing if they fit our purposes.

5.1.1 TextRunner

TextRunner [18] uses a *Single Pass Extractor* which tags all the sentences in a document with part-of-speech tags and noun phrase chunks as it goes. For each pair of noun phrases that are not too far apart, and subject to several other constraints, it applies a classifier to determine whether or not to extract a relationship.

The objective of the classifier is the element that decides whether a sequence of part-of-speech-tagged words is accurate enough to be a correct extraction or not. This element is trained using several heuristic constraints.

TextRunner does not offer a software implementation, and therefore we could not test it on our data.

5.1.2 Reverb

Reverb [5] is another Open IE system, which tries to correct the errors performed by other systems such as TextRunner.

According to the authors of Reverb, previous solutions had the following issues:

- Incoherent extractions: some of the relationships were no meaningful enough to con-

sider them as accurate. This problem is caused by the classifier, which chooses badly the words that form the relation phrase. Examples of this issue are the following:

Sentence	Incoherent Relation
The guide contains dead links and omits sites	contains omits
The Mark 14 was central to the torpedo scandal of the fleet	was central torpedo
They recalled that Nungesser began his career as a precinct leader	recalled began

Table 5.1: Examples of incoherent extractions

- Uninformative extractions: these relationships are those whose critical information has been omitted. This error is caused by improper handling of relation phrases that are expressed by a combination of a verb with a noun. Issues as described can be shown in the table below:

is	is an album by, is the author of
has	has a population of, has a Ph.D. in
made	made a deal with, made a promise to

Table 5.2: Uninformative relations (left) and their completions (right)

The authors of Reverb propose the use of constraints in order to improve the previous systems.

5.1.2.1 Syntactic Constraint

The first constraint proposed by Reverb is a syntactic constraint, limiting relation phrases to be a verb, a verb followed immediately by a preposition or a verb followed by nouns, adjectives or adverbs ending in a preposition.

In the event of various of these rules happening at the same time, the longest match is chosen, as it must be the one with the best information.

The use of this constraint prevents the extraction of incoherent tuples, and makes the system able to capture relation phrases expressed by a verb-noun combination.

5.1.2.2 Lexical Constraint

During the process, the system can extract some relation phrases that are too specific. The lexical constraint is used to separate valid relation phrases from over specified relation phrases, based on the intuition that a valid relation phrase should take many distinct argument in large corpus.

5.1.2.3 Limitations

Obviously, the use of the two constraints can limit the number of relation that can be extracted from a given corpora.

Nevertheless, the authors of Reverb assure that, on a set of sentences analysed, 85% of them satisfied both constraints. Therefore, although this system scope can never be 100%, it can be said that it is able (in theory) to capture a large amount of the relations contained in a text.

5.1.3 Ollie

Ollie [19] (Open Language Learning for Information Extraction) is a program that automatically identifies and extracts binary relationships from English sentences. Ollie is designed for Web-scale information extraction, where target relations are not specified in advance.

According to Ollie authors, Reverb (and other Open IE system) have two main problems:

- They can only extract relations mediated by verbs, or a subset of verbal patterns.
- They only perform a local analysis of the sentence.

Whereas ReVerb operates on flat sequences of tokens, Ollie works with the tree-like (graph with only small cycles) representation using Stanford's compression of the dependencies. This allows Ollie to capture expression that ReVerb misses, such as long-range relations.

These are some of the main characteristics of Ollie:

- Enabling conditions: An enabling condition is a condition that needs to be met for the extraction to be true. Certain words demark an enabling condition, such as "if" and "when". Ollie captures enabling conditions if they are present.

sentence: If I slept past noon, I'd be late for work.

extraction: (I; 'd be late for; work)[enabler=If I slept past noon]

- Attribution clauses: An attribution clause specifies an entity that asserted an extraction and a verb that specifies the expression. Ollie captures attributions if they are present.

sentence: Early astronomers believe that the earth is the center of the universe.

extraction: (the earth; is the center of; the universe)[attrib=Early astronomers believe]

- Relational nouns: Some relations are expressed without verbs. Ollie can capture these as well as verb-mediated relations.

sentence: Microsoft co-founder Bill Gates spoke at a conference on Monday.

extraction: (Bill Gates; be co-founder of; Microsoft)

- N-ary extractions: Often times similar relations will specify different aspects of the same event. Since Ollie captures long-range relations it can capture N-ary extractions by collapsing extractions where the relation phrase only differs by the preposition.

sentence: I learned that the 2012 Sasquatch music festival is scheduled for May 25th until May 28th.

extraction: (the 2012 Sasquatch music festival; is scheduled for; May 25th)

extraction: (the 2012 Sasquatch music festival; is scheduled until; May 28th)

nary: (the 2012 Sasquatch music festival; is scheduled; [for May 25th; to May 28th])

Nevertheless, as we can see later, Ollie is not capable of extracting unary relationships.

5.2 Experiments

After having studied different IE systems and their characteristics, it is time now to test them with our corpora.

In order to define a metric, we have chosen 50 random sentences that can be attributed to home automation processes. Afterwards, we have applied the selected systems on them. Results were verified by a human, to contrast the fidelity of the extractions.

We are going to study three major results of these tests:

- Precision : fraction of retrieved relations that are relevant.
- Recall : fraction of relevant relations that are retrieved.
- F_1 score: measure of the test accuracy that combines precision and recall.

In concordance to the objectives of this project, most of the sentences were in the imperative form; this is due to the fact that most of the phrases we are going to deal are not facts, but orders or steps to obtain a goal.

It is important to emphasize that the sentences in the text files should start with a capital letter, and end with a point. If not, Reverb will not detect them. Also, if you want to play with Reverb modifying the source code , you must set the locale of your system to en-US. You can do so in Java (will only affect the JVM) with the following line:

```
Locale.setDefault(new Locale("en", "US", "WIN"));
```

According to the results obtained, we have divided the extractions into three major groups:

- No extractions : the system under test has not been able to extract any relation on the sentence given.
- Illogical extraction : the IE system has provided some results on the sentence under test. Nevertheless, after visual inspection by a human, the extraction does not match the original meaning of the text.
- Good extraction : the system has made a good extraction, where the original meaning is in concordance with the result extraction. Further actions may be needed, such a Name Entity Recognition or article removal, among others.

With these groups identified, we can calculate precision, recall and F1-score follows:

$$Precision = \frac{GoodExtractions}{GoodExtractions + IllogicalExtractions}$$

$$Recall = \frac{GoodExtractions}{Sentences}$$

$$F_1 = 2 * \frac{Recall * Precision}{Recall + Precision}$$

All the tests were performed on the same computer, with the same files and Operating System. The experiments, unless otherwise specified, were done by using the default options, with the original source code (no modifications).

5.2.1 Reverb Results

The tests were made using the 1.3 version of Reverb. The results over 50 sentences were: 6 good extractions, 1 illogical extraction and 43 no extractions, as we can see in Fig. 5.1. Therefore,

$$Precision = \frac{6}{6+1} = 85,71\%$$

$$Recall = \frac{6}{50} = 12\%$$

$$F_1 = 2 * \frac{0.12*0.8571}{0.12+0.8571} = 21.05\%$$

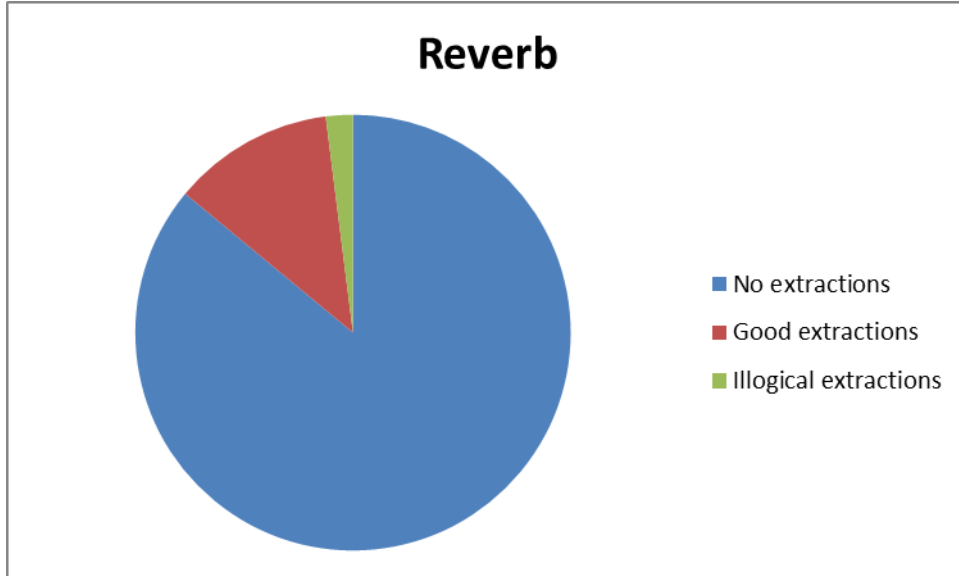


Figure 5.1: Summary of the results given by Reverb

On the following lines we will examine further these results.

5.2.1.1 Good extractions

As we can see, Reverb does not fit well with our corpora. Only 6 of the 50 sentences produced good results.

The sentences in this group are well structured, with the structure of a '*fact sentence*'. They are not imperatives or, if so, they contain a sub-sentence that can be identified as a fact sentence.

Examples are phrases as follow: *The TV is connected to the router.*, *Verify that the audio device transmits sound.* or *The router should be configured as an access point.*

5.2.1.2 Illogical extractions

The sentence *"Move to the directory you want to copy your files from"* produced as an extraction the triplet *(you ,want to copy ,your files)*.

It can be considered that the relation extracted is good, as it is one of the relations contained on that sentence. Nevertheless, we consider that the main relationship should be *(move, directory)* (an unary relationship) and therefore we have considered this as an illogical extraction.

5.2.1.3 No extractions

The biggest group is the one with the sentences which produced no relations.

As previously said, many of the test sentences were in the imperative form, and Reverb does not extract well relations from these kind of phrases. Examples of these sentences could be *"Link your computer to your smartphone"* or *"Unscrew the graphics card from the case"*.

Other sentences of our corpora follow the pattern of being unary relationships, that is, sentences where a change of the subject itself is declared, with no relations with other objects. Examples of this kind of sentences are *"Restart your computer"* or *"Turn on the TV"*.

Reverb can not extract unary relationships by default. Nevertheless, with the correct parameters defined, it can extract some of the relationships, as we will see later in this chapter, although further problems should be considered.

5.2.2 Ollie Results

In the execution of these tests, the version 1.0 of Ollie was used. The results obtained were: 18 good extractions, 7 illogical extractions and 26 no extractions, as can be seen in Fig. 5.2. Therefore,

$$Precision = \frac{18}{18+7} = 72\%$$

$$Recall = \frac{18}{51} = 35.3\%$$

$$F_1 = 2 * \frac{0.12 * 0.8571}{0.12 + 0.8571} = 47.37\%$$

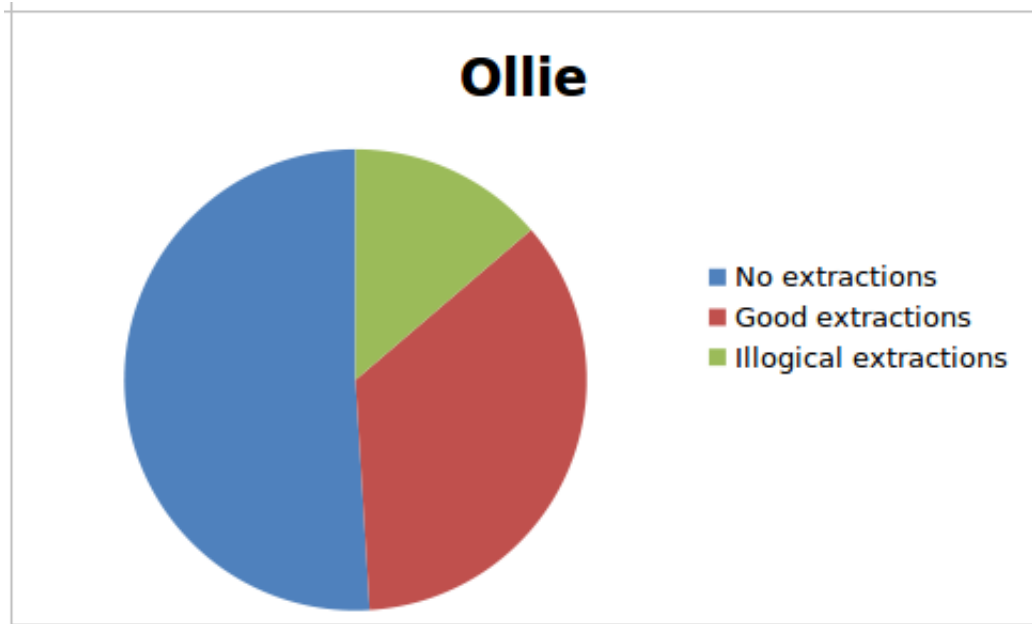


Figure 5.2: Summary of the results given by Ollie

Please note that the sum of these results is 51. This is due to the existence of n-ary extractions on Ollie, as noted before. On the following lines we will examine further these results.

5.2.2.1 Good extractions

Ollie performs way better than Reverb on the same corpora. It is a newer software, and the new characteristics available make a good difference when extracting relationships on our sentences. Nevertheless, less than half of the sentences produces extractions, which can be considered as a bad performance.

Many of the new extractions, in comparison to Reverb, are sentences in the imperative form. Sentences as "Screw the graphic card to the case" or "Verify that the audio device transmits sound" now produce satisfying extractions.

5.2.2.2 Illogical extractions

The number of total extractions delivered by Ollie is bigger than with Reverb. This also results in an increment in the number of illogical extractions. As an example, the sentence "Choose Image File from the menu" derived in the relation *(file, be image from, menu)*; or "Send an SMS to the service number" produced the triplet *(Send, be an SMS to, the service number)*. As we can see, the original sentences follow a common pattern, so the system could be trained to make good extractions from these sentences.

5.2.2.3 No extractions

The main reason why Ollie does not make extractions from sentences in our corpora is because they imply unary relationships, as in *Restart your computer* or *Launch your browser*.

5.2.3 Reverb with unary option results

While performing tests on the corpora, we realised that many of the relationships we should extract were unary relationships, where the sentences implied a change of some properties of the subject, with no other objects in the interactions.

A further research on this topic showed us that, with the following properties applied when running Reverb, this was capable of extracting unary relationships from sentences. We applied this method to the same corpora, and the results were: 13 good extractions, 17 illogical extractions and 20 no extractions, as seen in Fig. 5.3. Therefore,

$$Precision = \frac{13}{13+17} = 43.3\%$$

$$Recall = \frac{13}{51} = 25.5\%$$

$$F_1 = 2 * \frac{0.12 * 0.8571}{0.12 + 0.8571} = 32.1\%$$

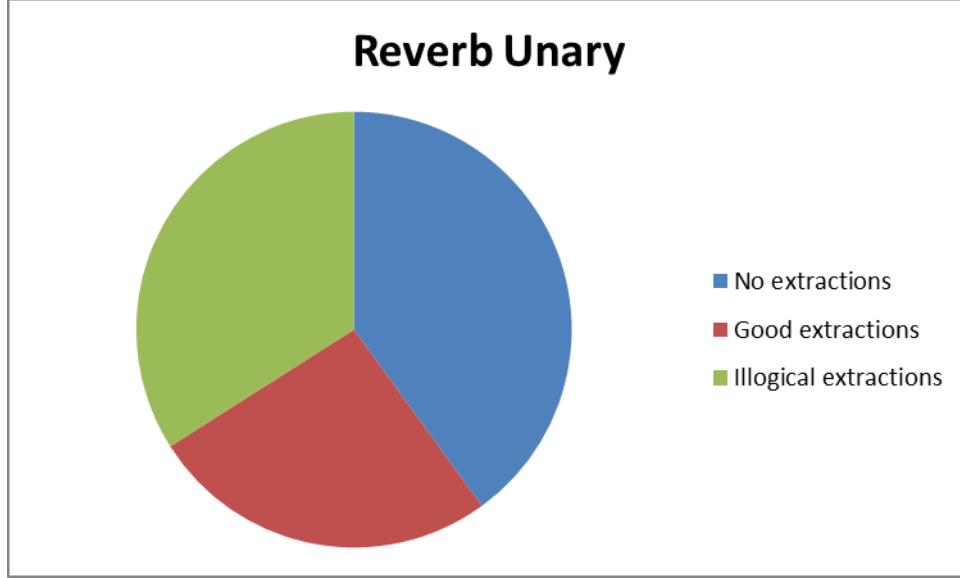


Figure 5.3: Summary of the results given by Reverb using the unary option

5.2.3.1 Good extractions

As imagined, the use of the unary option on Reverb produce more good relationships compared to the use of Reverb without options. In particular, sentences as *"The tv is switched on"* , *"Buy this circuit from a local retailer"* or *"Open a TERMINAL window"* produce satisfactory relationships with this new method.

5.2.3.2 Illogical extractions

In contrast to the good news on the satisfactory extractions, it has to be noted that the number of illogical extractions rose. In fact, the most worrying characteristic of this method is that some sentences that were previously well extracted , as *"Connect the phone to an USB cable"* , now produce illogical unary relationships (*connect the phone to, an USB cable*). As we see, it mistakes an imperative with an unary relationship.

5.2.3.3 No extractions

The no-extractions from this method are the same as with Reverb, with the exception of the unary relationships.

5.3 Conclusions

A summary of the results can be seen in Fig. 5.3.

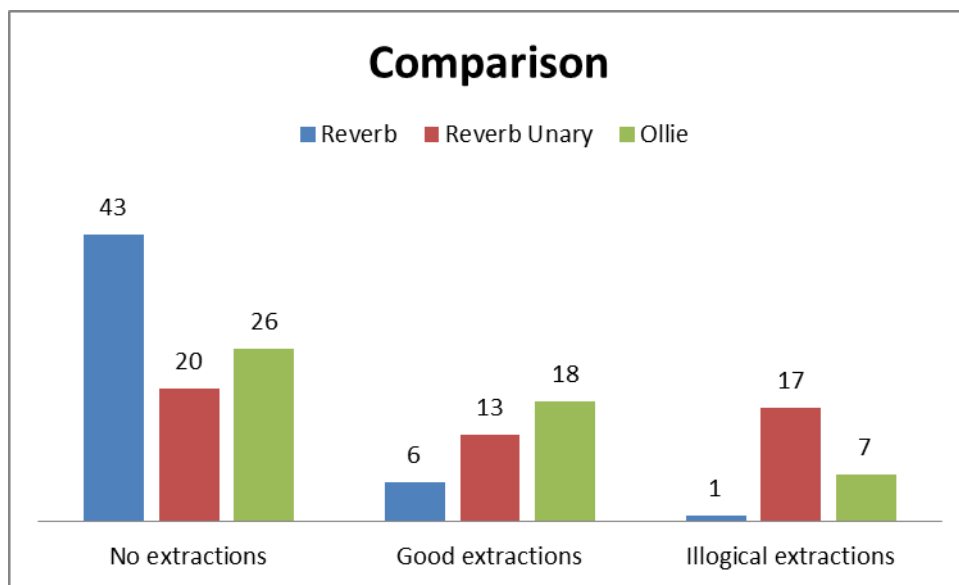


Figure 5.4: Summary of the results of the three systems

The results given demonstrate that there is not a obvious winner on the test performed with our corpora.

The characteristics of each method makes them suitable for some specific projects, but none of them produces good results with our corpus: the system admitting unary relationships does not accept imperatives forms, and so on.

It is clear that Reverb (with no extra options) is the system with the lowest performance, as we can see by its F1 score, but we couldn't decide which was better among the two others (Reverb with unary relationships, and Ollie).

At this point, a Solomonic decision was taken: we will use both of them, and test if the results were better than now. We will examine this option in the next section.

5.4 Information Extraction for Sensato

The Open Information Extraction system we have chosen for our project should combine the performance on imperative forms of Ollie, with the ability of extracting unary relationships of Reverb. Besides, we should take care that none of the good extractions lose their meaning after extracting unary relationships from them, as we saw on Sec. 5.3.

Therefore, the approach we have taken on our project is:

- Run Ollie on our corpora.
- Remove the sentences that has already given an extraction (logical or illogical, as we can not determine this automatically).
- Run Reverb with the unary relationships option on the remaining sentences.
- Join the results given by the two systems.

After performing all these steps, the experiment produced the following results: 28 good extractions, 7 illogical extractions and 16 no extractions, as can be seen in the Fig. 5.5. Therefore,

$$Precision = \frac{28}{28+7} = 80\%$$

$$Recall = \frac{28}{51} = 55\%$$

$$F_1 = 2 * \frac{0.12 * 0.8571}{0.12 + 0.8571} = 65.18\%$$

The performance of this mixed system is much better than the previous ones. This hypothesis is validated by its precision and recall indicators (which are among the best) and its F1 score (which is the best). Although it can not provide all the extractions, it provides almost all of the relationships from simple sentences, even if they are in imperative form or implies unary relationships. Sentences with complicated structures provide no extractions.

In the Fig. 5.6 we can see a comparison among all the systems analysed.

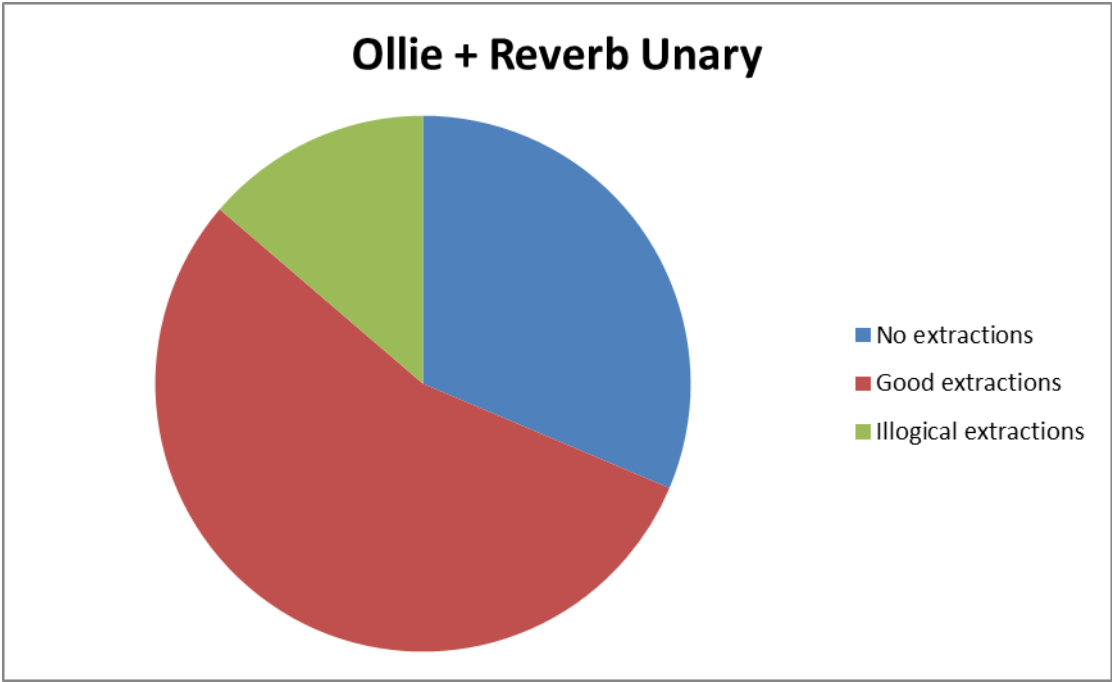


Figure 5.5: Summary of the results of the new system

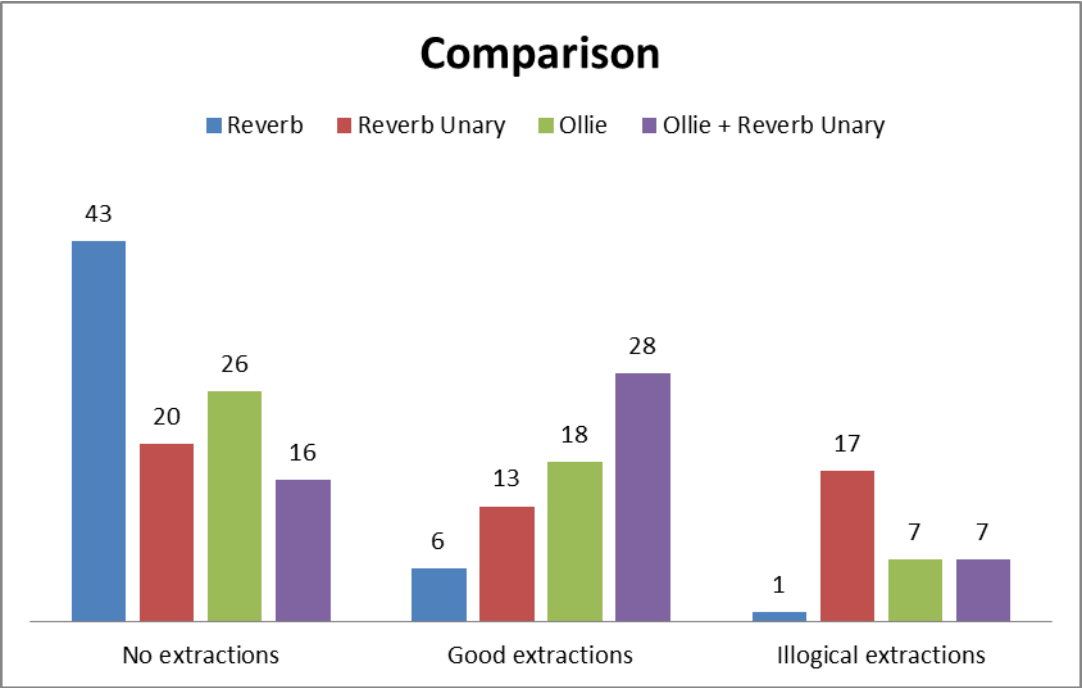


Figure 5.6: Comparison of all the systems analysed

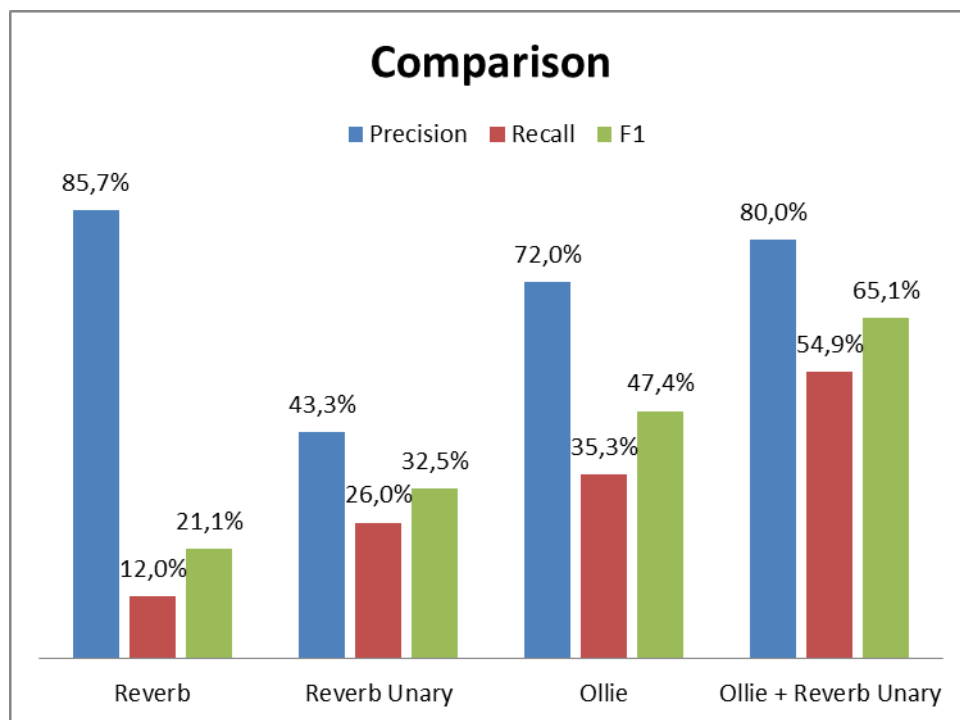


Figure 5.7: Summary of the precision, recall and F-score of all the system analysed

Knowledge Adaptation for Sensato

6.1 Introduction

In chapters 4 and 5 we have described how our system was capable of collecting knowledge used in smart home automation. The next step was creating a system capable of adding this data to a knowledge base, in order to make it more intelligent; also, we will use this platform for bringing the user a list of tasks he should perform in order to achieve some goal in a Smart Home Automation environment.

6.2 Knowledge base

The knowledge base used in Sensato is ConceptNet, which we have previously discussed in Sec.2.2. ConceptNet is flexible enough to fit our purposes, and it is also extremely powerful when working with a lot of data.

As we did not want to modify the existing knowledge available on ConceptNet, we decided to make a branch, running a local version of ConceptNet on a known server. The steps in order to do so are explained in Appendix B

6.2.1 Running the API Server

In order to access the data we have added from other applications, we have to set up an API. ConceptNet provides a REST API that can be deployed via a WSGI file, which is a simple and universal interface between web servers and web applications or frameworks for the Python programming language.

There are three methods for accessing data through the ConceptNet 5.1 API: lookup, search, and association.

- **Lookup:** find the edges that include a known URI from an object.
- **Search:** find a list of edges that match certain criteria.
- **Association:** find concepts similar to a particular concept or a list of concepts.

For the development of our project, the *'search'* method will be the most used.

6.3 Inserting Smart Home Automation knowledge in ConceptNet

In chapters 4 and 5 it has been discussed how the Smart Home Automation knowledge was retrieved. Nevertheless, the information at that point was only "plain text", not able to be indexed, and therefore it is not suitable for making searches or relating it with other available data.

In order to add this knowledge into ConceptNet, we have to structure it in a special way, so that it can be added as stated in Appendix B.2 . In this subsections we will discuss the approach that has been taken, and the technologies used, with the final goal of getting a JSON file with all the knowledge retrieved in previous steps, with the structure of edges.

6.3.1 JSON

The files with the information we want to add should be structured as a JSON file.

JSON stands for JavaScript Object Notation , and it is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. An example JSON file could be as the following:

```
person = {  
  "first": "John",  
  "last": "Doe",  
  "age": 39,  
  "sex": "M",  
  "salary": 70000,  
  "registered": true,  
  "interests": [ "Reading", "Mountain Biking", "Hacking" ]  
}
```

As we can see, it describes an 'object' with multiple attributes, separated by commas. The value of this attributes can be easily retrieved as it is a 'key/value' format; a simple key can have multiple values, as the '*interests*' key in this example.

6.3.2 Edges

As stated in Sec. 2.2.1, an edge represents the information between two nodes. Therefore, we have to construct all the edges with the Smart Home Automation that we have, in order to completely add the knowledge that we have obtained.

A ConceptNet edge is represented as a mapping (dictionary) of the following fields:

- **id**: the unique ID for this edge, which contains a SHA-1 hash of the information that makes it unique.
- **uri**: the URI of the assertion being expressed. The uri is not necessarily unique, because many edges can bundle together to express the same assertion.
- **rel**: the URI of the predicate (relation) of this assertion.
- **start**: the URI of the first argument of the assertion.
- **end**: the URI of the second argument of the assertion.
- **weight**: the strength with which this edge expresses this assertion. A typical weight is 1, but weights can be higher, lower, or even negative.
- **sources**: the sources that, when combined, say that this assertion should be true (or not true, if the weight is negative).
- **license**: a URI representing the Creative Commons license that governs this data. See Copying and sharing ConceptNet.
- **dataset**: a URI representing the dataset, or the batch of data from a particular source that created this edge.
- **context**: the URI of the context in which this statement is said to be true.
- **features**: a list of three identifiers for features, which are essentially assertions with one of their three components missing. These can be useful in machine learning for inferring missing data.
- **surfaceText**: the original natural language text that expressed this statement. May be null, because not every statement was derived from natural language input. The locations of the start and end concepts will be marked by surrounding them with double brackets. An example of a surfaceText is "[[a cat]] is [[an animal]]".

The process of building the edges has been done with a Python program that we have coded. We have made use of some of the ConceptNet code, which was needed in order to obtain some fields as the id, the weight or the URI. Other fields have been filled with our information, such as *rel* , *start* or *end*.

Finally, a JSON file containing the edges is similar to the following

```
"add": {
  "doc": {
    "endLemmas": "the DVD player",
    "license": "GSI",
    "context": "/ctx/all",
    "end": "/c/en/the DVD player",
    "features": [
      "/c/en/The dvd is inserted in -",
      "/c/en/The dvd - /c/en/the DVD player",
      "- is inserted in /c/en/the DVD player"
    ],
    "weight": 1.0,
    "start": "/c/en/The dvd",
    "startLemmas": "The dvd",
    "uri": "/a/[is inserted in/,/c/en/The dvd/,/c/en/the DVD player/]",
    "dataset": "GSI",
    "sources": "/and[/GSI/l/,/GSI/u/]",
    "rel": "is inserted in",
    "id": "/e/d50c30a65b131e5dd073dbb3c072f4148a0d9b0d"
  },
  "boost": 1.0
},
"add": {
  "doc": {
    "endLemmas": "the router",
    "license": "GSI",
    ...
  }
}
```

For the development of this project, we needed to insert the knowledge into our Knowledge Base in two different ways:

- As a list of steps : Our system is aimed to give the user the needed steps in order to complete a task in a home automation process. Therefore, the information should be inserted as a list of steps, in order to be able to retrieve them.
- As concepts : Our system should help the user when one concept is unknown for him. If we insert the information retrieved as concepts, we will be able to relate them with other concepts, or find previous knowledge about them in the information already extracted by ConceptNet (or other Knowledge Bases).

6.3.3 Knowledge as a list of steps

The list of steps needed to perform a Smart Home Automation Task should be added in a special way.

The entry point of this stage is the files obtained in the knowledge acquisition module , seen on Sec. 4. That is, we have one file for each Smart Home Automation task, with all the steps needed in order to complete it.

It was decided to add these steps as prerequisites, starting from the top. Therefore, we should start with the final objective of the task (i.e, "*Watch the tablet on the TV*"), and then we should list a prerequisites to perform that task. This way, all the steps will be chained, until the first step is added.

The fields when filling the edges (Sec. 6.3.2) should be as the following:

- startLemmas: the n-th step of the task.
- endLemmas: the (n-1)-th step of the task.
- rel: */relation/HasPrerequisite*, a relation from ConceptNet useful to indicate prerequisites.

6.3.4 Knowledge as concepts

This is the usual way of inserting knowledge into a knowledge base. By inserting our retrieved information as concepts, the nodes in the graph would be entities (i.e: TV, router, DVD player...) and the edges represent the link between these concepts (for example: inserted in, connected to...).

In this case, we must fill the fields when creating the edges (Sec. 6.3.2) as follow:

- startLemmas: the first argument extracted by our IE System.
- endLemmas: the second argument extracted by our IE System.
- rel: the relation extracted by our IE System.

Therefore, a JSON file with this format of knowledge should be as the presented in Sec. 6.3.2.

In the special case of unary relationships, we will use a special relationship in order to insert it into our knowledge base, so the edges should be completed as the following:

- startLemmas: the first argument or the second argument (whichever exists) extracted by our IE System.
- endLemmas: the relation extract by our IE System.
- rel: */relation/HasProperty*, a relation from ConceptNet useful to indicate properties or abilities of the concepts.

The final JSON file with the edges in this format should be as this:

```
{
  "add": {
    "doc": {
      "endLemmas": "Turn on",
      "license": "GSI",
      "context": "/ctx/all",
      "end": "/c/en/Turn on",
      "features": [
        "/c/en/TV /relation/HasProperty -",
        "/c/en/TV - /c/en/Turn on",
        "- /relation/HasProperty /c/en/Turn on"
      ],
      "weight": 1.0,
      "start": "/c/en/TV",
      "startLemmas": "TV",
      "uri": "/a[/relation/HasProperty/,/c/en/TV/,/c/en/Turn on]",
      "dataset": "GSI",
    }
  }
}
```

```
    "sources": "/and[/GSI/l/,/GSI/u/]",
    "rel": "/relation/HasProperty",
    "id": "/e/89dcc30cd12650e40afc395623e21d61a95d639a"
  },
  "boost": 1.0
},
"commit": {}
}
```


Interaction Agent for Sensato

7.1 Introduction

At this point, we have already explained how we obtain information about how to perform a Smart Home task (Sec. 4) and how we extract knowledge and relations from it (Sec. 5).

In this chapter, we will explain how we interact with the final user (which could be a person, or another piece of software) to present them this information when needed.

7.2 Use Cases

First of all, we had to ask ourselves how would a typical user usually interact with the knowledge that we have inserted as stated in Sec. 6. After an analysis of the situation, we have identified two main use cases.

7.2.1 Use case 1

In the first use case identified, the user wants to know the steps needed to perform some Home Automation task.

Therefore, the user asks the agent how to achieve some goal, and it should answer with all the needed steps in order. The illustration of this use case can be seen in Fig. 7.1.

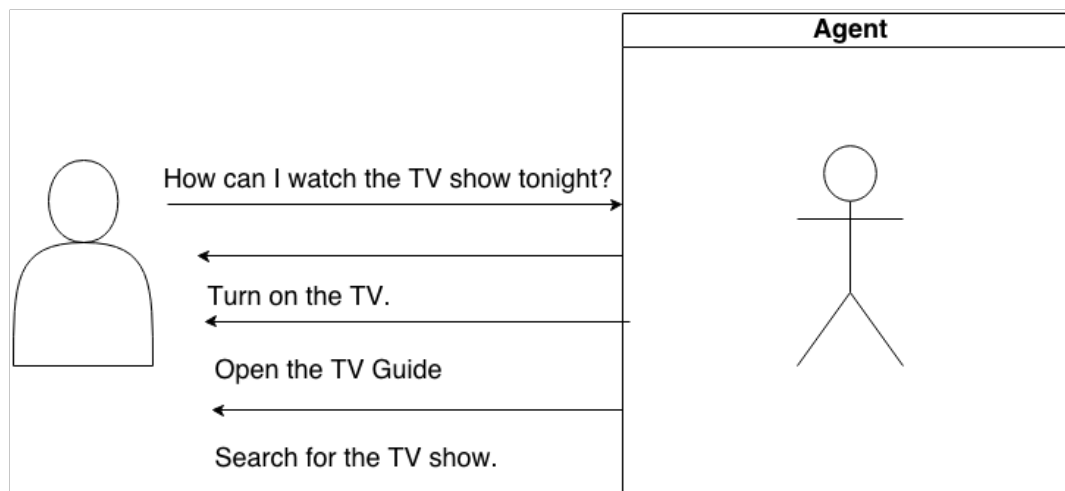


Figure 7.1: First use case of the system.

7.2.2 Use case 2

The second use case we have identified is the following: the user is stuck on some step, and he wants to know further information about some Home Automation object.

For a lot of people, concepts as 'router', 'HDMI cable' or even 'WiFi' are not easy to understand. If we are going to ask them to perform some tasks with that objects, it seems like a good idea to help them to identify those objects, and what are they used for.

Therefore, the user may ask to our system about what is some object, or what is it used for. The illustration of this use case can be seen in Fig. 7.2.

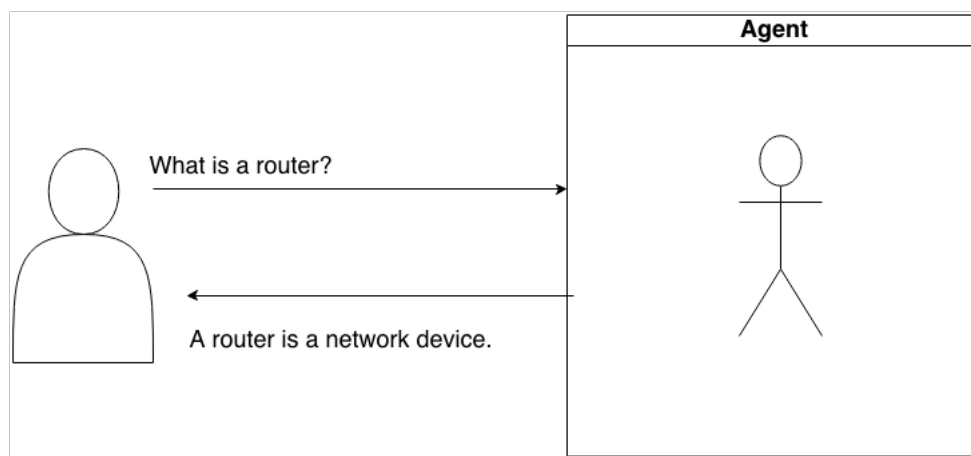


Figure 7.2: Second use case of the system.

7.3 Implementation of the agent

Conversational agents are human-like agents with the ability to create linguistic interaction by using engines that analyse the user input (i.e. what the user writes) and find and fetch the best possible answer consulting their knowledge base and sometimes fetching more related information from the environment. For the purposes of this project, and in order to be able to perform the use cases described, we have developed a chatterbot, which is a computer program designed to simulate an intelligent conversation with one or more human users.

7.3.1 Structure of the agent

The agent developed in this project had to be able to answer different kinds of questions, and retrieving data from the knowledge base in different forms. It also had to interact with

the backend, and with the final user. Therefore, we had to structure the agents in different modules, as seen in Fig. 7.3.

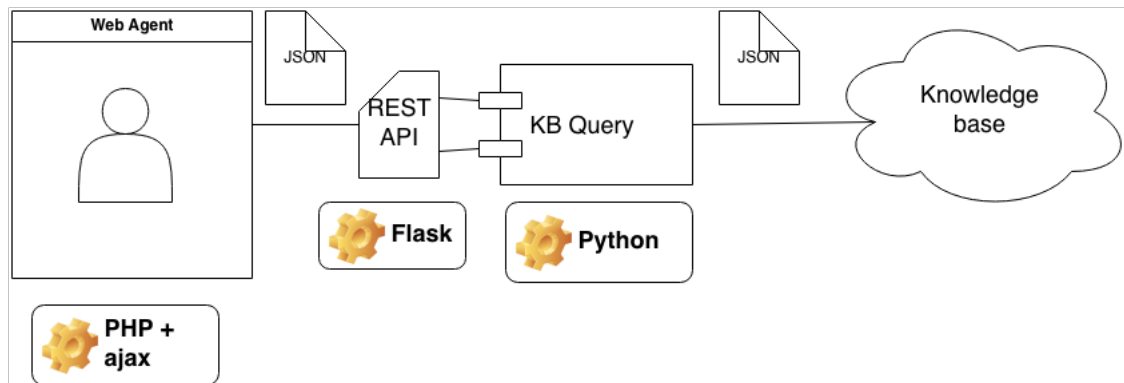


Figure 7.3: Structure of the agent

In the following lines we will discuss the details of each one of the modules.


7.3.1.1 KB Query

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

In this module, we develop a Python piece of software that is capable of making request to the knowledge base based on the input of the program. The first step is to identify what kind of request is the user making, based on the input of the user. To do so, we search for keywords in the petition that can help us identify the user's intentions. Then, we make an appropriate request to the knowledge base. As we have previously seen, the information from ConceptNet is retrieved in JSON format, so we have to parse it. A sample run of the program can be seen on Fig 7.4. In that figure, we see that, when asked for some task, the module can retrieve all the needed steps to perform it. It can also retrieve data of possible uses of some objects, and definitions of them.

In the final steps of the development of this module, we decided to add a new functionality: it will retrieve images from Wikipedia ¹ when asked about some concept. This way, the final answer could be more complete, and in next steps we could show visual information to the users.

¹http://en.wikipedia.org/wiki/Main_Page



```

alejandro@alejandro-desktop: ~/Sensato
alejandro@alejandro-de... x alejandro@alejandro-de... x alejandro@alejandro-de... x alejandro@alejandro-de... x alejandro@alejandro-de... x
alejandro@alejandro-desktop:~/Sensato$ python main.py http://127.0.0.1:8084 WatchAFilmInTheTV
Option 1: turn on the tablet
Option 2: download the movie from a torrent site
Which option would you like to take? TurnOnTheTablet
Option 1: connect the tablet to the tv
Which option would you like to take? ConnectTheTabletToTheTV
Option 1: launch google play movies
Option 2: launch netflix
Which option would you like to take? LaunchNetflix
You have completed the task. Congratulations
alejandro@alejandro-desktop:~/Sensato$ █

```

Figure 7.4: Running the Python module

In order to do so, we downloaded a Python implementation of a Wikipedia API from <https://github.com/goldsmith/Wikipedia>. It allowed us to retrieve the pictures from Wikipedia articles easily.

Finally, the output of the program is a JSON file containing all the information previously discussed: the knowledge obtained from the knowledge base about the user's request, and a image that could help the user to identify the concepts. A sample output of our program could be as the following (supposing that the input is *'what do you use hdmi for?'*):

```

{
  "img": "http://upload.wikimedia.org/wikipedia/commons/8/85/
    Adapter_dvi_hdmi_S7302224_wp.jpg",
  "text": [
    "connecting devices in order to transmit audio and video"
  ]
}

```

We chose Python because its easiness. Also, with Python we could import libraries such as *json* or *urllib2* that will help us to achieve our goals.

7.3.1.2 RESTful API

The next step in the development of the agent was creating an interface between our web-based agent and the Python module. As our agent is going to be web-based, we decided to implement a RESTful API in the Python module.

A RESTful API [20] (also called a RESTful web service) is a web API implemented using HTTP and REST principles. It is a collection of resources, with four defined aspects:

- the base URI for the web API, such as `http://example.com/resources/`
- the Internet media type of the data supported by the web API. This is often JSON but can be any other valid Internet media type provided that it is a valid hypertext standard.
- the set of operations supported by the web API using HTTP methods (e.g., GET, PUT, POST, or DELETE).
- The API must be hypertext driven.

In order to create the RESTful API, we decided to use Flask [21]. Flask is a BSD-licensed microframework for Python based on Werkzeug² and Jinja 2³. With Flask, we have *built-in-the-box* RESTful request dispatching, so it was easy and convenient to use it in our project.

To integrate it with our module, we had to decide which routes (URLs) were allowed by our system, and define which method will run our module when the user access this URLs. For the purposes of our project, we define three valid routes.

- `http://localhost:5000/whatuses/<concept>` : when this URL is called, the module asks the knowledge base about all possible uses of *concept*.
- `http://localhost:5000/whatis/<concept>` : A search about the meaning of *concept* is done on the knowledge base.
- `http://localhost:5000/stepbefore/<step>` : The system searches for the following step in a Home Automation Process, based on the actual step.

²<http://werkzeug.pocoo.org/>

³<http://jinja.pocoo.org/docs/>

7.3.1.3 Web Agent

The last module is a web-based agent that receives the inputs from the user, and shows the information that the previous modules have retrieved.

For the development of this module, we use the technologies that in were discussed in Sec.4.3, PHP and ajax. With PHP we make the requests to the RESTful API, and with ajax we show them to the user.

In Fig. 7.5 , a sample request to the agent about a concept is shown. In Fig. 7.6, the user asks for the possible uses of HDMI. Finally, in Fig.7.7 , we can see the agent showing the user the possible next steps in a Home Automation Task.



Figure 7.5: Agent answering what HDMI is.

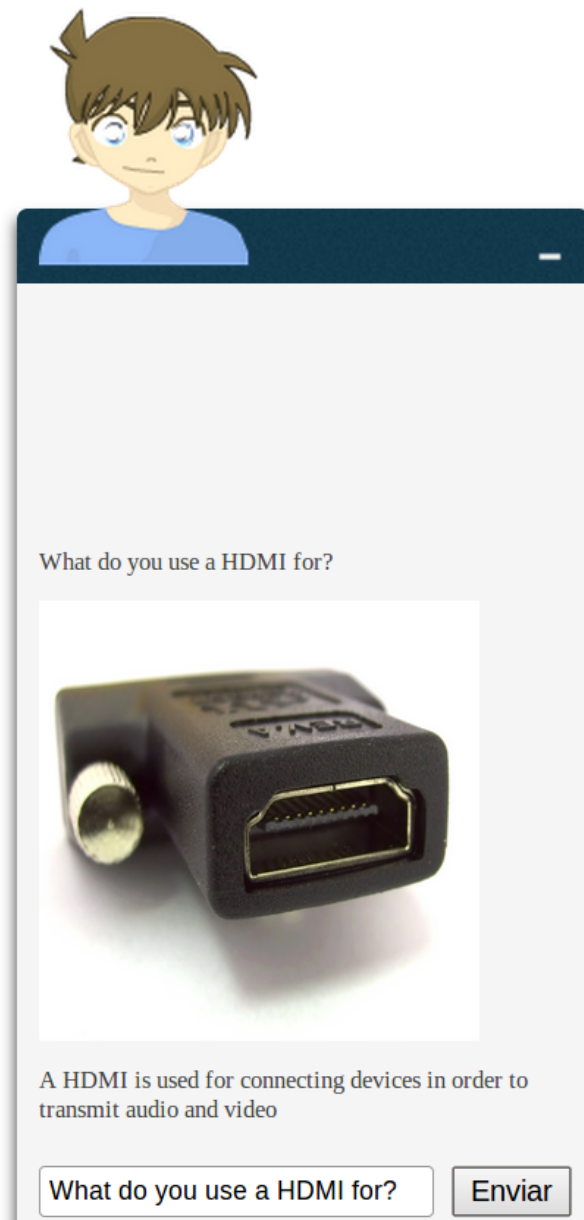


Figure 7.6: Agent answering what HDMI is used for.



Figure 7.7: Agent helping the user to perform a Home Automation Task

Conclusions and Future Work

8.1 Introduction

In this chapter, we comment the achieved goals and we expose the conclusions deduced after implementing this project. Finally, we present some different possible lines of investigation oriented to improve the system developed within this project.

The project aimed to develop a whole system capable of learning and using Smart Home Knowledge by using Common Sense. This project includes the knowledge acquisition and extraction stages, as well as the knowledge adaptation into a knowledge base and its presentation and interaction with the user via an agent. In the following lines we will discuss if that stages have been correctly implemented, as well as future work for improving them.

8.2 Conclusions

At this point, with all the modules implemented correctly, we integrated them together and tested how they work. As a brief abstract, we could highlight the following points.

- The knowledge acquisition system do its job correctly, and it is a good system for acquiring the knowledge that we aimed. Nevertheless, the number of users participating in the project has not been as numerous as desired, so the knowledge acquired has not been big enough in order to be representative. Therefore, we will later discuss some of the solutions that may solve this problem in Sec. 8.3.1 and Sec. 8.3.2.
- The information extraction module was the stage on which we had the most problems, so a lot of research was needed. At first, the singularity of our corpora led us to believe that none of the information extraction services available suited our corpus; nevertheless, after struggling with this problem, the proposed solution (a mix between Ollie and a special version of ReVerb) seems to perform quite well.
- The knowledge adaptation module is fully functional. ConceptNet fit our purposes quite well, and its integration with other modules has been quite easy, thanks to the JSON format file. Due to the variety of relations present in ConceptNet, we could add all the information retrieved from previous steps easily, and in a standardized way. The main problem in this stage was running a local version of ConceptNet, which was quite difficult. This point is explained in Appendix B.
- The agent developed in order to interact with the final user was a chatterbot. It achieves the scope within this project, as it is capable of helping the user with Home

Automation tasks, helping him with complementary information when needed. It answers correctly when asked questions that follow a predetermined pattern, but it is not capable of maintaining a full conversation in natural language. Therefore, some improvements are proposed on Sec.8.3.4 and 8.3.5

8.3 Future Work

This project is delivered as a fully functional prototype, and all the goals determined in the scope of the project have been achieved. Nevertheless, when implementing some functionalities, we realised that there might be (even) better ways to perform them, or different approaches that may be interesting for research. In the following sections we will discuss some of them, with the hope that some of them could be integrated into our project some day.

8.3.1 Integrate the Knowledge Acquisition module in Social Networks

The use of social network has increased dramatically in the last few years, and, nowadays, its penetration in society is huge, as can be seen on Fig. 8.1.

Some of these social network, such as Facebook¹ or Tuenti², provide an API in order to create applications within its platform. We can use the big database of users in these social network (and the spare time that they users have) to our advantage.

From our point of view, the ideal platform to develop a fork of our knowledge acquisition system is Facebook. It provides a complete API to interact with their database, and they have a huge amount of users that enter on Facebook daily. What is more, most of the applications on Facebook are PHP-based, so the integration of our system should be simple. Facebook have a website dedicated to developers, with much further information, on <https://developers.facebook.com/>

The only disadvantage is that it requires a server with a SSL certificate to host the application, and that may be expensive if only used for research purposes.

¹www.facebook.com

²www.tuenti.com

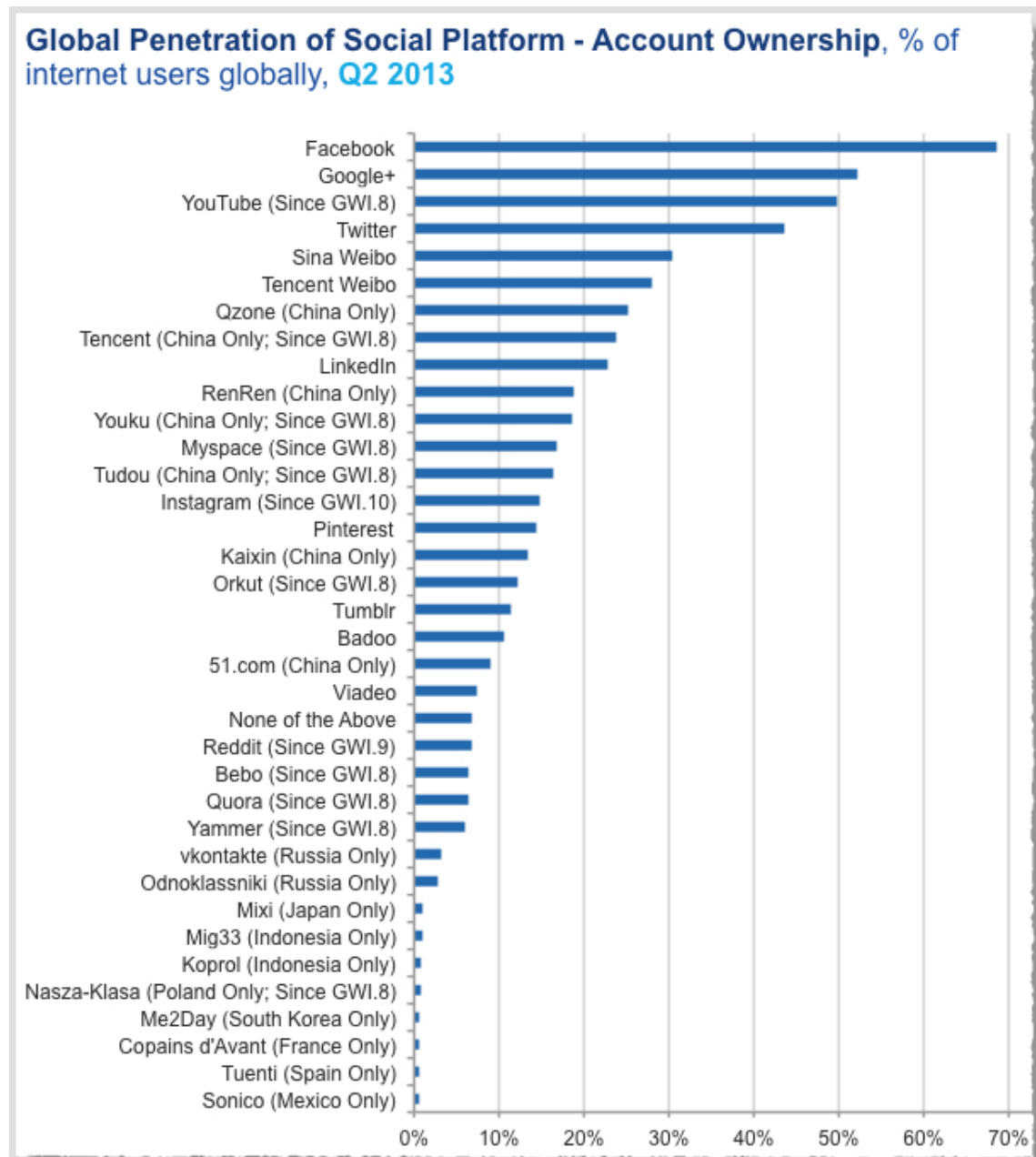


Figure 8.1: Penetration of social platforms.

8.3.2 Acquire knowledge from websites

The approach taken in this project was to acquire the knowledge via a crowd-sourced game. Nevertheless, there are many webpages that already have this kind of knowledge (although that it may not be as accurate). Therefore, in order to increase the knowledge that we adapt in our knowledge base, it could be a good idea to acquire knowledge from these websites.

In order to do so, a scrapper of this website should be developed. Once it is done, it could be easily integrated in our project. The information retrieved by this scrapper should also go through the Information Extraction subsystem, in order to extract more concepts and relationships from this source.

From example, we can see what steps are suggested by WikiHow in order to connect an Android tablet to a WiFi network in Fig. 8.2.

8.3.3 Better connection between different kind of knowledges

As we commented in Sec. 6.3.3 and Sec.6.3.4 , we have inserted two different kinds of knowledge in our knowledge base. For the development of this project, some of this knowledge was connected (for example, the concepts that are implicit in some of the steps). Nevertheless, this is not a fully automated process, and some research should be done in order to connect them, and take advantage from it.

8.3.4 Integrating the agent with Jade/Jadex/Wade

At this moment, we have only one agent running on a known server. JADE [22](Java Agent DEvelopment Framework) simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 1.4 of JAVA (the run time environment or the JDK). A JADE Agent Platform can be seen in Fig. 8.3.

Therefore, in the future, we could create a JADE agent with similar functionality than ours. The main problem may be is that our agent is developed mostly in Python, while

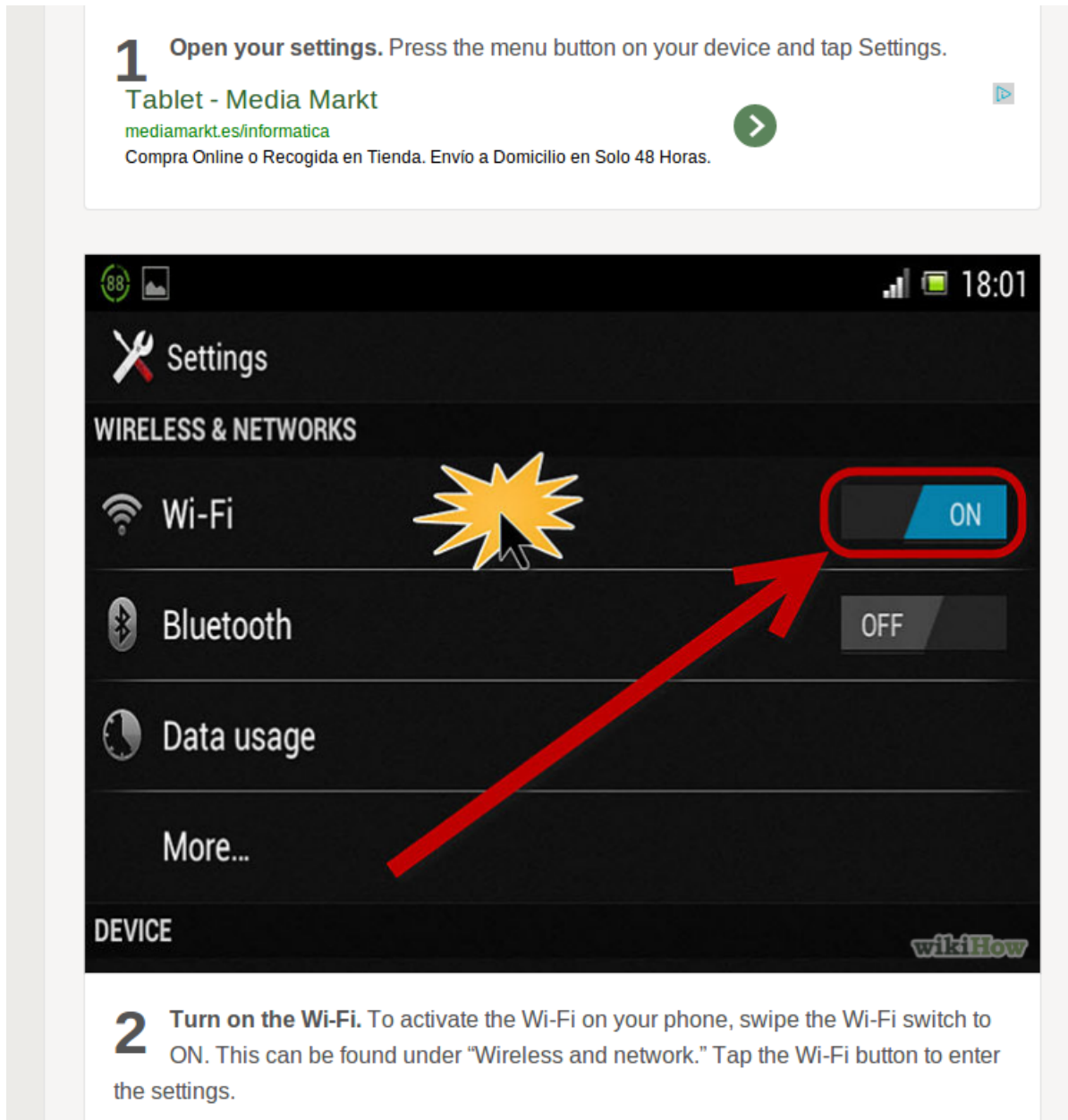


Figure 8.2: Steps about a Smart Home Automation task by wikiHow.

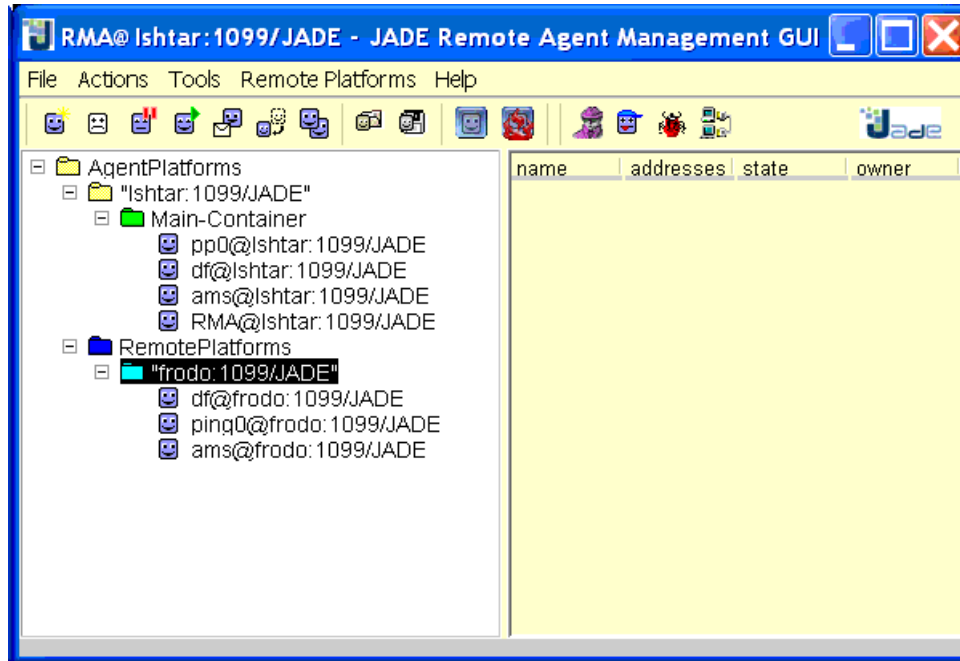


Figure 8.3: JADE Agents platform.

JADE uses Java. Nevertheless, with a JADE agent we get more advantages, such as the possibility of using WADE [23] (that provides support for the execution of tasks defined according to the workflow metaphor).

8.3.5 Using NLP techniques in the agent

As previously mentioned, our agent answers to some question with a predetermined pattern. If we want it to establish a conversation with a human, we should use some NLP [24] (Natural Language Processing) techniques on it.

Natural language processing (NLP) is a field of computer science, artificial intelligence, and linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human–computer interaction. Many challenges in NLP involve natural language understanding; that is, enabling computers to derive meaning from human or natural language input.

Several NLP toolkits are available online in order to apply NLP techniques. Among them, we recommend to use NLTK [25], as it is fully compatible with Python (the same programming language that we have used in most of the development of our project).

8.3.6 Integrating the web-agent with other systems

In the research group, several other Agents have been implemented in the last years. Due to the technology used, we think that it may be a good idea to integrate our project with the Personal Agent developed in Javier Herrera's master thesis [26].

In this thesis, a Personal Agent is developed using Natural Language Processing Techniques. It uses SIREn [27], which is an entity retrieval system designed to provide entity search capabilities over datasets as large as the entire Web of Data; and Bottle [28] as the front-end controller. A complete architecture of the system can be seen in Fig. 8.4.

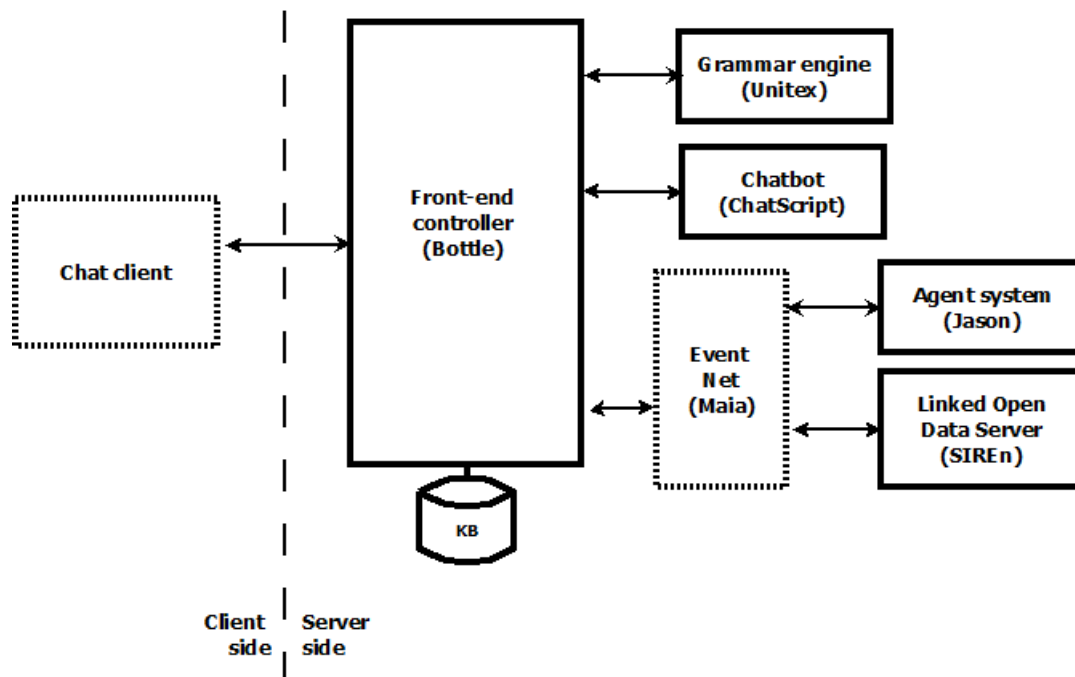


Figure 8.4: Architecture of personal agent solution.

Tests sentences

During the development of this project, we had to choose a group of sentences from our corpora in order to test the IE systems. In the following pages these sentences can be viewed, for independent experiments or other uses. These phrases were chosen from technical manuals, Web Pages (i.e WikiHow) and other similar sources, and are intended to be representative of our corpora.

A.1 Test sentences

The tv is switched on.

Connect the router to the TV.

Insert the DVD into the DVD player.

Turn on the TV.

Make sure that the TV is on.

The dvd is inserted in the DVD player.

Select the first option of the menu.

Make sure that the TV has been connected to the router.

Change the configuration of your router into Access Point.

The router should be configured as an access point.

The USB Flash Drive should be inserted to the laptop.

Switch on the phone's bluetooth.

Set the TV into HDMI mode.

Download the app into cell phone.

Connect the phone to an USB cable.

Link your computer to your smartphone.

Adjust the volume of your earphones.

Launch your browser.

Install the software into your computer.

On the menu, select the third option.

Do double click on the icon.

Disconnect all internal cables connected to the graphics hardware.

Unscrew the graphics card from the case.

Screw the graphic card to the case.

Input your data in a spreadsheet.

Install the CPU into the motherboard.

The tv is connected to the router.

Connect the wires from the front panel of the computer case to the motherboard front panel pins.

Choose "Image File" from the menu.

Test the connection to MySQL database.

Drag the icon into the trash.

Verify that the audio device transmits sound.

Set the headset for stereo audio.

Draw the circuit layout on the copper coated board.

Press the Enter key on your keyboard .

Wire the primary winding of the transformer to the main AC supply.

Attach the other side of the wire to the other side of the battery.

Make a phone call to a known number.

Write an email to the technical support.

Buy this circuit from a local retailer.

Keep local copies of everything.

Burn your data in a DVD.

Restart your computer.

Contact your Internet Service Provider.

Move to the directory you want to copy your files from.

Establish a connection to the internet.

To delete the whole row, type dd in that row.

Open a TERMINAL window.

Bring your computer to warranty service.

Send an SMS to the service number.

Installing a local version of ConceptNet

For the purposes of this project, we had to run a local version of ConceptNet 5 on a home server. This process was not simple at all, as it is not well documented, and there are some hard-coded IPs which we had to find all over the source code. This appendix aims to be an initial point for those who would like to run a local version of ConceptNet.

B.1 Installation process

The process of installing a local version of ConceptNet is not trivial, and it required the use of different technologies that are described on the following lines.

We assume that the PC is running a Debian-based Linux distribution (for example, the test were done on Ubuntu 13.04), and that it Python 2.7.4 (or newer) and Apache 2 are installed. If not, many manuals are available on the Net, and the instructions are easy to follow.

B.1.1 Setting up the enviroment

First, we have to install some of the packages and dependencies of the ConceptNet software.

The module *wsgi* of Apache is needed to support server-side Python applications on Apache. As most of the ConceptNet code is on Python, the installation of this module is required.

To do so, we must open up a console and type the following lines:

```
sudo apt-get install libapache2-mod-wsgi
```

Then, we need to download some Python packages required for ConceptNet.

Once again, we should go to a console, and type:

```
sudo apt-get install python-pip python-dev build-essential git
pip install Flask
pip install simplenlp
pip install ipython
sudo pip install virtualenv
sudo pip install -e git+https://github.com/commonsense/metanl.git#egg=metanl
sudo aptitude install libc6-dev python-dev python-virtualenv
sudo apt-get install gunicorn
```

B.1.2 Installing Solr

Solr is an open source enterprise search platform from the Apache Lucene project which provides highly scalable distributed search and index replication.

Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Apache Tomcat or Jetty. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it usable from most popular programming languages. Solr's powerful external configuration allows it to be tailored to many types of application without Java coding, and it has a plugin architecture to support more advanced customization.

ConceptNet uses Solr in order to index all the edges and lemmas available.

For the development of the project, we should download Solr from <http://conceptnet5.media.mit.edu/downloads/20120501/conceptnet5-solr-config.tar.gz>. Then, it should be unzipped on any folder (if possible, in the home folder of the user).

B.1.3 Installing ConceptNet

The first step is downloading ConceptNet. In this project we have used the version 5, but any newer version should work. The download link is <https://github.com/commonsense/conceptnet5/archive/master.zip>.

Then, we must unzip it in the `/var/www` folder of our system. Root permission may be needed to do so.

Now, we must change some hardcoded IPs in the system. They are in the file `'api.py'`, under the folder `'/var/www/conceptnet5-master/conceptnet'`.

The lines to be changed are the line 153 and 158. In both, we must change the URL to fit with our personal Solr IPs (which can be shown by running the `java.jar` file on Solr).

For example, in our project, these lines were as follow:

```
153: params['shards']='mudarra.gsi.dit.upm.es:8983/solr'
```

```
158: SOLR_BASE ='http://mudarra.gsi.dit.upm.es:8983/solr/select?'
```

B.2 Adding the knowledge from ConceptNet

Once the previous steps have been done, we have the platform running, but there is no knowledge contained on it. For the purposes of our thesis, we had to add the information that was already retrieved by the authors of ConceptNet.

In order to do so, there is file available on its website (http://conceptnet5.media.mit.edu/downloads/current/conceptnet5_solr_json_20130917.tar.bz) which is pre-formatted for being added to Solr.

Once the file is downloaded, adding it to our local system is easy: a shell script program (available on the Solr package) should be run, and it will be added .

If all the previous steps have been done correctly, our local version of ConceptNet should be now installed. In Fig. B.1 we can see an example of a search query of the 'car' concept.



Figure B.1: ConceptNet running on a local server

Sorting algorithm

In the development of this project we had to create an algorithm to sort all the valid steps provided by an user. In the following pages the source code is shown, for informative purposes

```

import java.util.ArrayList;

/**
 * Class with the aux methods in order to merge arrays of sentences.
 *
 * @author alopez
 *
 */
public class MergeFiles {

/**
 * This method looks at the relative positions of the candidate sentences
 * to be added
 * (i.e, the sentences in the most advanced position in each ArrayList)
 * and return
 * the index of the array which next sentence is going to be added
 * To do so, it adds each row of the relativePositions array, and it
 * to add the one
 * with the minimum (i.e, the one that appears the less in other files,
 * or the one which is
 * nearer to the present position
 * @param relativePositions
 * @return the index of the array which next sentence is going to be added
 */
private static int fileOfSentenceToAdd(int[][] relativePositions){
int [] sum = new int[relativePositions.length];
int min=0;
int file=0;

for(int i = 0; i< relativePositions.length;i++){
for(int j = 0; j<relativePositions[i].length;j++){
sum[i]+=relativePositions[i][j];
}
if(i==0) min=sum[0];
else{
if (sum[i]<=min){
min=sum[i];

```

```
file=i;
}
}
}
return file;
}
/**
 * Returns a bi-dimensional array with the relative positions of the
 * sentences in each other arraylist
 * For example, row 1 represents the relative position of the sentence
 * the first arraylist.
 * Then , relativePosition[0][0] is equal to 0 (the sentence is located
 * in the actual position).
 * If a sentence is not found in some other arraylist, it is filled with
 * a -1
 * For example, if the sentence in the first arraylist does not appear in
 * the second arraylist, relativePosition[0][1] is equal to -1
 * @param sentences
 * @param position
 * @param files
 * @return
 */
private static int[][] relativePositions(String[] sentences,int[] position,
                                         ArrayList<String> ... files){
int[][] relativePositions = new int[sentences.length][files.length];
for(int i = 0; i< sentences.length;i++){
for(int j = 0; j< files.length;j++){
if(!files[j].contains(sentences[i])) relativePositions[i][j]=-1;
else{
relativePositions[i][j] = files[j].indexOf(sentences[i])
- position[j];
}
System.out.println(relativePositions[i][j]);
}

}
return relativePositions;
```

```

}

/**
 * Takes a number of arraylist, and executes an algorithm in order to create
 * a 'merge' of all of them, keeping the order
 * IMPORTANT: the last sentence in each arraylist should be the same in
 * order to work correctly.
 * For example, the last sentence could be "THE END" or something similar.
 * @param Unknown number of String ArrayList with the sentences of each file
 * @return Merged arraylist
 */
public static ArrayList<String> mergeSentences(ArrayList<String> ... files){

    int numberOfFiles = files.length;
    int[] position = new int[numberOfFiles]; //It shows how much we have moved
    //inside each arraylist
    String[] sentencesInPosition = new String[numberOfFiles];
    ArrayList<String> merged = new ArrayList<String>();

    /* Get the first element of each arraylist */
    for(int i=0;i<files.length;i++){
        sentencesInPosition[i] = files[i].get(position[i]);
    }

    while(true){
        for(int i=0;i<files.length;i++){
            sentencesInPosition[i] = files[i].get(position[i]);
        }
        int toAdd =fileOfSentenceToAdd(relativePositions(sentencesInPosition,
            position,files));
        String stringToAdd =files[toAdd].get(position[toAdd]);
        merged.add(stringToAdd);
        for(int i=0;i<files.length;i++){
            if(files[i].contains(stringToAdd)){
                if(files[i].indexOf(stringToAdd)+1 <files[i].size()){
                    position[i]=files[i].indexOf(stringToAdd)+1;
                }
            }
        }
    }
}

```

```
}else{
position[i]=files[i].size()-1;
}
System.out.println("posicion " + i +" es igual a " +position[i]);

}
}
System.out.println("");
boolean end = true;
for(int i=0;i<files.length;i++){
if(position[i]!=files[i].size()-1){
end=false;
}
}
if(end)break;
}
return merged;
}
}
```

Bibliography

- [1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.
- [2] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.
- [3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.
- [4] Hugo Liu and Push Singh. Conceptnet—a practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226, 2004.
- [5] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011.
- [6] Tom O Hara. Overview of Cyc. *Technology*, (February), 2002.
- [7] Luis Von Ahn, Mihir Kedia, and Manuel Blum. Verbosity: a game for collecting common-sense facts. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 75–78. ACM, 2006.
- [8] Henry Lieberman, Dustin Smith, and Alea Teeters. Common consensus: a web-based game for collecting commonsense goals. In *ACM Workshop on Common Sense for Intelligent Interfaces*, 2007.
- [9] Robert Speer, Jayant Krishnamurthy, Catherine Havasi, Dustin Smith, Henry Lieberman, and Kenneth Arnold. An interface for targeted collection of common sense knowledge using a mixture model. In *Proceedings of the 14th international conference on Intelligent user interfaces*, pages 137–146. ACM, 2009.

- [10] Push Singh, Thomas Lin, Erik T Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open mind common sense: Knowledge acquisition from the general public. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 1223–1237. Springer, 2002.
- [11] Social. WiredHowTo. http://howto.wired.com/wiki/Main_Page. [Online; accessed november 2013].
- [12] Social. WikiHow. <http://www.wikihow.com/Main-Page>. [Online; accessed november 2013].
- [13] Jesus Castagnetto, Harish Rawat, Sascha Schumann, Chris Scollo, and Deepak Veliath. *Professional PHP programming*. Wrox Press, 1999.
- [14] UV Ramana and TV Prabhakar. Some experiments with the performance of lamp architecture. In *Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on*, pages 916–920. IEEE, 2005.
- [15] Jerry Lee Ford Jr. *Ajax programming for the absolute beginner*. Course Technology Press, 2008.
- [16] Social. Gamification: How Effective Is It? <http://www.slideshare.net/ervler/gamification-how-effective-is-it>, 2011. [Online; accessed november 2013].
- [17] Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.
- [18] Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007.
- [19] Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. Open language learning for information extraction. In *Proceedings of Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CONLL)*, 2012.
- [20] Leonard Richardson and Sam Ruby. *RESTful web services*. O’Reilly, 2008.
- [21] Social. Flask. <http://flask.pocoo.org/>. [Online; accessed november 2013].

- [22] F Bellifemine G Caire, A Poggi, and G Rimassa. Jade. a white paper, 2003.
- [23] Federico Bergenti, Giovanni Caire, and Danilo Gotta. Interactive workflows with wade. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*, pages 10–15. IEEE, 2012.
- [24] Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.
- [25] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.
- [26] Javier Herrera. Design and implementation of a personal agent architecture based on bot technologies and information retrieval. 2012.
- [27] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice. com: a document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.
- [28] M Hellkamp. Bottle-python web framework, 2012.

