INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Situated Dialogue for Speaking Robots

Eugénio Alves Ribeiro

Dissertation submitted to obtain the Master Degree in
Information Systems and Computer Engineering

Jury

President:	Doctor José Carlos Alves Pereira Monteiro
Supervisor:	Doctor David Manuel Martins de Matos
Member:	Doctor Rodrigo Martins de Matos Ventura

October 2012

Acknowledgements

First of all, I would like to thank everyone who spent some precious time answering the multiple inquiries, as their results took a very important role on our evaluation and decisions.

Furthermore, I would like to thank Artica[1] for providing us the Magabot[2] we used to develop our prototype and Filipa Menezes for giving us the opportunity to present our prototype at the Game On[3] exposition.

Additionally, I would like to thank my advisor, David Martins de Matos, for all the patience and clever advice.

Also, I would like to thank all of my friends who kept on asking things about my thesis, even when not understanding any part of it.

Finally, I would like to thank my parents, Isabel and Jorge, as well as my girlfriend, Beatriz, without whom I would not be where I am today.

Thank you all!

Lisboa, November 22, 2012
Eugénio Alves Ribeiro

[1] http://artica.cc
[2] http://magabot.cc
[3] http://gameon.gameover.sapo.pt/

It never ends...

Resumo

Este documento apresenta uma *framework* para o desenvolvimento de robots capazes de aprender através do diálogo. Inicialmente são apresentados a motivação, os problemas e os objectivos do trabalho, assim como algum trabalho relacionado nas diferentes áreas cobertas pelo nosso projecto, que pode ser útil no desenvolvimento deste tipo de robots. Sistemas de diálogo, agentes inteligentes e representação do conhecimento são algumas das áreas abrangidas.

Em seguida é apresentada a arquitectura da *framework*, uma arquitectura funcional, com três camadas, estando a mente no topo, as competências no meio e o corpo no fundo. Esta arquitectura foi desenvolvida com foco na extensibilidade, flexibilidade e portabilidade.

Seguidamente são apresentados os nossos dois cenários de teste, assim como a nossa implementação da arquitectura para esses cenários. O primeiro cenário involve um robot capaz de aprender diferentes cores e identificá-las posteriormente, enquanto o segundo involve um robot capaz de conhecer pessoas e lembrar-se dos seus nomes.

Por fim, apresentamos a avaliação desses dois cenários, tanto objectivamente como subjectivamente, assim como as conclusões que fomos capazes de extrair dos resultados alcançados e algumas sugestões para trabalho futuro.

Tendo os resultados da avaliação sido positivos, a framework revelou-se útil, pelo menos em cenários simples.

Abstract

This document presents a framework for the development of robots able to learn through dialogue. It starts by presenting the motivation, problems and objectives of the work, as well as some related work in the different areas our project covers which might be useful when developing this kind of robots. Dialogue systems, intelligent agents and knowledge representation are some of the areas covered.

Following, the framework's architecture is presented. It is a functional, layered architecture, with the mind on the top layer, competences on the middle layer and the body on the bottom layer. This architecture focuses on extensibility, flexibility and portability.

After that we present our two test scenarios and our deployment of the architecture for those scenarios. The first scenario involves a robot able to learn different colors and identify them later while the second involves a robot able to meet people and remember their names.

Finally, we present the evaluation of those two test scenarios, both objective and subjective, as well as the conclusions we were able to extract from the results and future work suggestions.

As the evaluation had positive results, the framework was proved useful, at least in simple scenarios.

Palavras Chave
Keywords

Palavras Chave

Sistemas de diálogo
Agentes inteligente
Aprendizagem
Base de conhecimento
Raciocínio
Robots

Keywords

Dialogue systems
Intelligent agents
Learning
Knowledge base
Reasoning
Robots

Index

List of Figures

1 Introduction

"A long time ago, in a galaxy far, far away..."
– Star Wars

1.1 Motivation

Some years ago, artificial agents that could interact with and learn from the surrounding environment through their senses were part of science fiction. Some good examples of this are the famous robots from the Star Wars saga, R2D2 and C3P0, which are still affectionately remembered nowadays by a large amount of people, some of them who did not even watch the movies.

Characters like these affected people's minds, causing a desire for their existence in real life. This, in turn, led to a major increase of research in the field of artificial intelligence, which is still going on nowadays. Autonomous agents and multi-agent systems, speech and language processing, vision processing, robotics and many other fields have evolved so much that, today, we still do not have robots like the ones from Star Wars but we can say that, in the future, we probably will.

This thesis's objective is to advance some steps further into that reality, by developing a framework that allows agents to obtain knowledge from the surrounding environment through dialogue and acting according to that knowledge. This is an ambitious project but any advance that can be made through it will be very important in this difficult path that AI is following and we hope that some time in the future we will be able to look back and proudly believe that our work has been important in the development of agents which open so many doors and can help mankind in ways we probably are not even aware of nowadays.

1.2 Expectations

Although we thought that developing a framework that enables robotic agents to learn through dialogue was a very good idea, we were not sure if the people in general shared the same opinion. Because of this, before starting the development of such framework, we performed an inquiry to make sure that the idea was welcome by the people in general and to find out the expectations they had from such kind of robots. The obtained results will now be presented.

From the 115 people inquired, 23 were female while 92 were male and their ages ranged from 12 to 55 years. First of all, there were some background questions from which we could extract that 99% of the inquired people use the computer at least twice a week, with Microsoft Windows being by far the most popular operative system, followed by Mac OS on second place

and any kind of Linux distribution on the last place, as can be seen on figure 1.1. Also, 64% of the inquired have at least one smartphone or tablet, mostly with Android as operative system, followed by iOS, as we can see on figure 1.2.

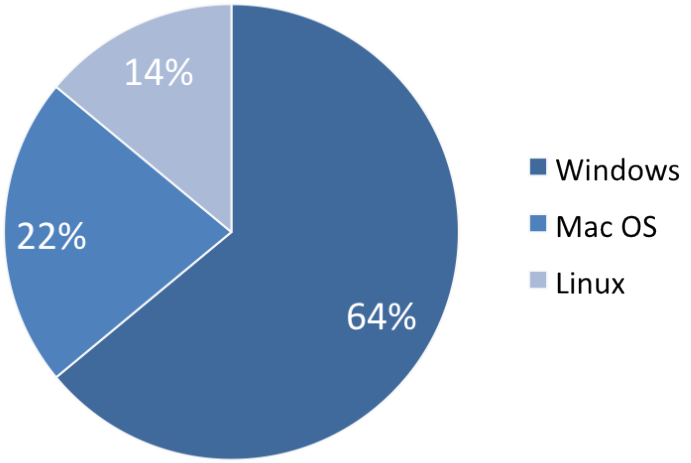

Figure 1.1: People's favorite operative system

When entering the robotics domain, only 13% of the inquired have at least one robot, however, as we can see on figure 1.3, the majority of the ones who answered affirmatively have built their own robots, which supports the community theory presented on chapter 1.

Moving on to the questions truly related to our project, when prompted what was their first though after reading 'robots that learn through dialogue', the inquired were divided between 'Cool!' and 'That is the future!', with 33 and 37% of the answers respectively, as can be seen on figure 1.4. However, 14% were more skeptical and answered 'Scary...' or 'Is that possible?'. Nonetheless, only 9% of the inquired believe that such robots would not be useful to mankind, alleging, on the one hand, that they would either destroy us instead of helping us or aggravate the economical crisis by occupying jobs that actually need human action or, on the other hand, that they would never be able to do what a man can do and be limited forever.

As for the other 91%, they expect these robots to be able to perform multiple tasks, especially the repetitive and tiring ones humans typically do not want to do, like housekeeping. Also, people expect these robots to substantially improve quality of life by sparing humans from doing these kinds of tasks, protecting and saving lives, for example on fires, and providing social support for the elderly, sick or handicapped people, amongst other possible scenarios.

Furthermore, people believe that on the one hand, these robots could specialize on one subject, forming a community of highly specialized robots, while on the other hand they could obtain knowledge about multiple subjects, functioning as individuals able to perform multiple tasks.

There are also some people that expect such agents to be important information sources able to teach other agents, both human and artificial.

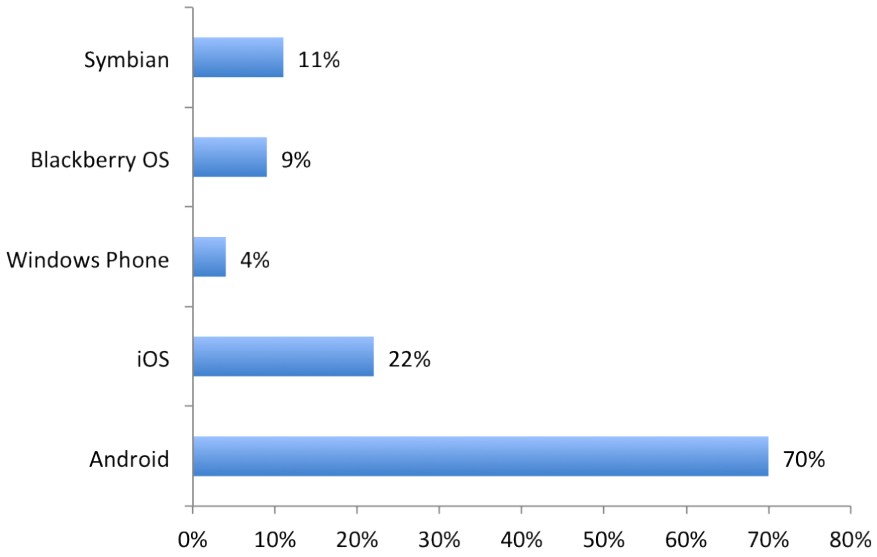

Figure 1.2: Most used mobile operative systems

As usual, ethical questions are the most controversial, as although most of the inquired think that these robots should always obey humans and, otherwise, would be scared of and totally against them, there are some people who think that they should have free will, at least at the same level that humans have.

After obtaining these results, we became even more motivated to continue our work. However, we also became aware of some risks and ethical questions we had to consider and were not aware before.

1.3 Requirements

For a robotic agent to be able to learn through dialogue it must combine dialogue management, knowledge representation and reasoning, action planning and body control. Furthermore, as we want it to learn properties of objects present in the surrounding environment and be able to act accordingly, it must also include sensors able to capture those properties, with vision sensors being the most common.

Given this, we defined the following general requirements for this kind of agents:

- The agent must be able to maintain a coherent dialogue.
- The agent must be able to identify different elements in the environment.
- Talking about unknown elements in the environment should help the agent obtaining knowledge about it.
- The agent must be able to improve its actions and complete tasks according to its knowledge.
- The system must have a body, which or whose properties might change over time.

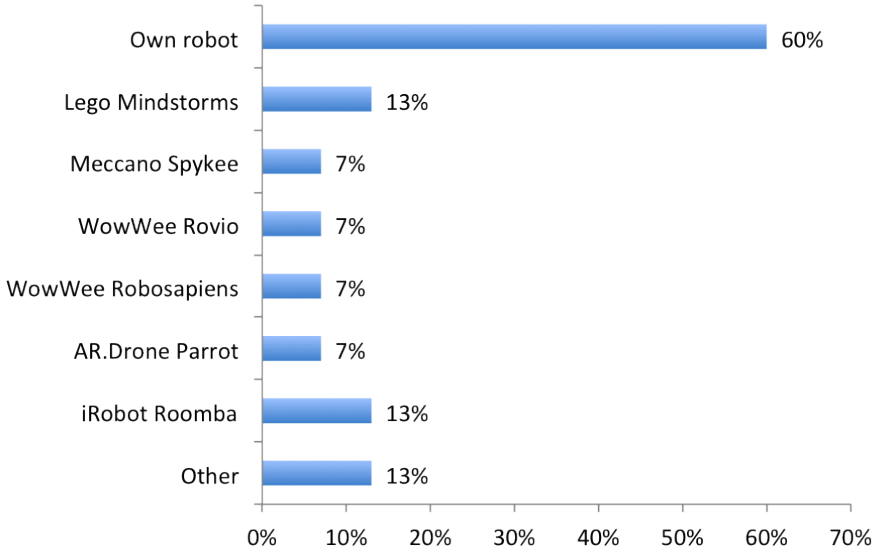

Figure 1.3: Robots owned by the inquired people

1.4 Problems

Enabling a robot to learn things about the surrounding environment through dialogue places many challenges, such as:

- Communicating with other agents (human or not) in the environment;
- Identifying elements in the environment;
- Obtaining knowledge from and about the environment;
- Reasoning about the obtained and previously existing knowledge;
- Solving knowledge conflicts;
- Acting according to the obtained knowledge.

1.5 Objectives

As for this thesis work, we defined as objective the development of a theoretical framework that could help in the development of robots able to learn through dialogue.

This framework must enable the creation of such robots, without limiting their functionality and adaptability to multiple scenarios.

It must define some high level processes for the mapping between language and knowledge, as well as for action planning adaptation according to the existing knowledge and possible actions.

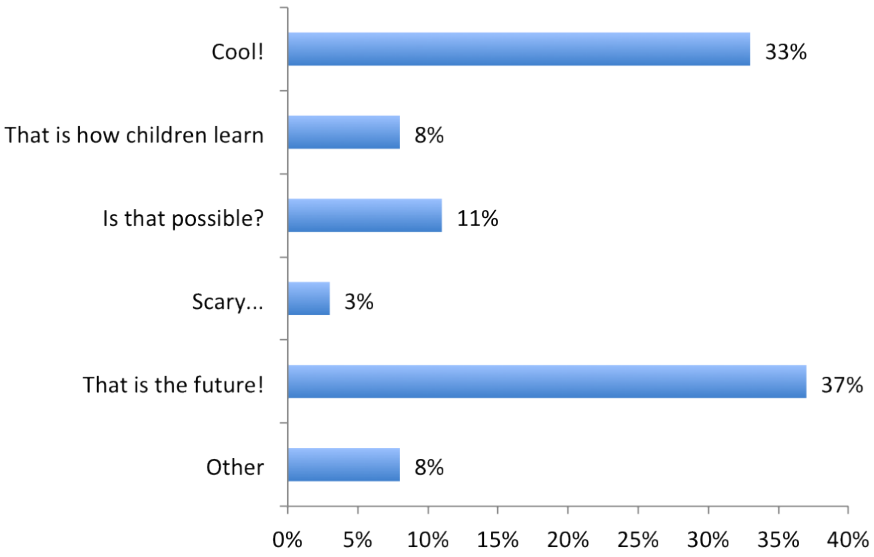

Figure 1.4: People's first thought after reading 'robots that learn through dialogue'

Furthermore, it must provide simple adaptability to multiple and different physical or virtual bodies.

In addition to the theoretical framework, at least one test scenario must be defined and a prototype implementation of the framework for that scenario must be developed for a matter of testing and validity checking.

1.6 Document Structure

This document presents the previously referred framework, explaining the choices made and criticizing the obtained results. It is organized as follows:

- Chapter 2 - *Related Work* - gives an overview of the state of the art in the areas related to this thesis' work.
- Chapter 3 - *Architecture* - presents the architecture of the system, explaining the multiple system components and the relations between them.
- Chapter 4 - *Deployment* - presents our implementation of the previously presented architecture, for our two test scenarios.
- Chapter 5 - *Evaluation* - describes the evaluation process of the system and presents the outcoming results.
- Chapter 6 - *Conclusions* - gives a final overview on the work done and proposes some future work to improve the system.

2 Related Work

"No. I am your father."
– Darth Vader in Star Wars Episode V: The Empire Strikes Back

2.1 Definitions

As this thesis aggregates components from distinct and complex AI areas, such as agent systems, natural language processing, knowledge representation and reasoning and robotics, it is important that the reader is familiar with the following definitions, as the terms are widely used in the work presented bellow.

2.1.1 Intelligent Agent

An intelligent agent is a computer system that is situated in some environment, in which it is capable of autonomous action, in order to meet its design objectives (Wooldridge 2002).

2.1.2 Learning Agent

A learning agent is an agent that can improve its performance through learning (Russell and Norvig 2002). Learning is very important because the effort needed for programming complex intelligent machines by hand is much larger than the one for teaching those machines. Also, we tend to create agents inspired in humans or animals, for which learning is intrinsic.

2.1.3 Dialogue System

A dialogue system or conversational agent is an agent that can interact with other agents, human or not, using some kind of human language, such as text, speech, emotions or gestures (Teng, Liu, and Ren 2009), typically across multiple turns or sentences.

2.1.4 Knowledge Base

A knowledge base is a special kind of database for knowledge management, providing means for computerized storage, organization, and retrieval of knowledge (Akerkar and Sajja 2010).

2.1.5 Ontology

An ontology is a specification of a conceptualization or, more specifically, a description of concepts and relationships about a certain subject (Gruber 1993). Ontologies are used for knowledge storage, sharing and reuse, which makes them very useful for intelligent agents, when they perform tasks that involve reasoning about some specific knowledge.

2.2 Dialogue Processing

Dialogue is one of the main focuses of our project. However, it does not focus on how speech is recognized or synthesized but is rather concerned with how to update the knowledge of the environment through dialogue, even when the environment and the dialogue are open.

In this section we describe some existing frameworks for building dialogue systems, which can help us fulfill our requirements. Although we will need speech recognizers and synthesizers we will not present them because, as stated before, their functionalities are not focused by our project and, thus, we can use any, as long as they have acceptable performance.

2.2.1 TRIPS - The Rochester Interactive Planning System

TRIPS (Allen, Ferguson, and Stent 2001) is a conversational system that can interact robustly and almost in real-time using spoken language and other modalities. According to its authors, it has successfully engaged in dialogues with untrained users in different simple problem solving domains. However, its first version had some major limitations.

The first identified limitation was due to the enforcing of strict turn taking and sequential processing of each utterance through three stages - interpretation, dialogue management and generation. This kind of restriction makes the interaction less natural for humans, as it does not permit frequent components of human-human dialogues, such as the grounding of each other's contributions, acknowledgements, pauses and interruptions. This limitation is present in many dialogue systems, as it is very difficult to overcome.

Another limitation was the lack of initiative due to system behavior being driven by the dialogue manager, which focuses on interpreting user input, neglecting the system's own independent goals. A natural conversational agent's behavior is determined by the combination of the last utterance, its own goals and other external events it becomes aware of, and not just by the first one.

The last major limitation identified by TRIPS's authors was the lack of portability caused by the direct use of domain and task-specific knowledge in the dialogue manager.

To address these problems, the architecture shown on figure 2.1 was developed. This architecture defines a core for the conversational system which supports asynchronous interpretation, generation, planning and acting. Also, it has a clear separation between discourse modeling and task/domain levels of reasoning. This design simplifies the incremental development of new behaviors, enhances the system's ability to handle complex domains, improves portability and allows task-level initiative. The three main components, representing interpretation, generation and behavior, are the following:

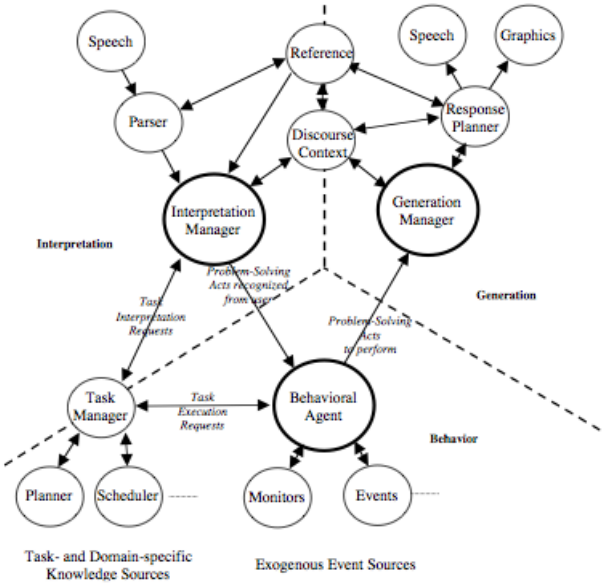

Figure 2.1: TRIPS Core Architecture (Allen, Ferguson, and Stent 2001)

2.2.1.1 Interpretation Manager

The interpretation manager interprets incoming parsed utterances, updating the discourse context. It starts by producing turn-taking information, which is a simple task when using a push-to-talk interface, but can be very difficult if using an open-mike interface. To deal with this problem, TRIPS interprets the input incrementally, as the recognized constituents come from the parser. However, the interpretation manager's main task is identifying the intended speech act, the problem solving act that it triggers and the obligations arising from the interaction.

2.2.1.2 Generation Manager

The generation manager generates balanced discourse plans based on the received problem solving goals and discourse obligations. As it operates asynchronously from the other components, it can be continuously planning, even without new information from the interpretation and behavior modules, by using timing information.

For simple interactions, such as groundings and greetings, the generation manager does not need the problem solving state, as it uses simple rules based on adjacency pairs. However, sometimes it needs to ask for information from the behavioral agent to satisfy discourse obligations, or can receive goals from it which can be planned to be satisfied even without discourse obligations. Also, the generation manager can plan extensive discourse contributions using rhetorical relations.

2.2.1.3 Behavioral Agent

The behavioral agent is responsible for the system's overall problem solving behavior. This behavior is defined as a function of the problem solving acts produced by the interpretation

manager, the goals and obligations of the system, and external events the behavioral agent becomes aware of. As previously noted, all these factors must be taken into account and not just the first one. When faced with an event the behavioral agent decides whether to simply communicate it or take initiative and to what level. Also, it must decide whether to be cooperative or not.

2.2.1.4 Conclusions

TRIPS deals with problems and limitations that most dialogue systems do not even care about. Its component separation would be an example to follow, simply for permitting easy extension and great portability between domains. However, its major importance comes from its contribution for a much more natural conversational agent. Although our project's main focus is the ability to learn from the dialogue, we want that dialogue to be as natural as possible and, in that matter, we can learn and benefit a lot from the effort put in TRIPS by its authors.

2.2.2 Galatea

Galatea (Kawamoto, Shimodaira, Nitta, Nishimoto, Nakamura, Itou, Morishima, Yotsukura, Kai, Lee, Yamashita, Kobayashi, Tokuda, Hirose, Minematsu, Yamada, Den, Utsuro, and Sagayama 2002; Kawamoto, Shimodaira, Nitta, Nishimoto, Nakamura, Itou, Morishima, Yotsukura, Kai, Lee, Yamashita, Kobayashi, Tokuda, Hirose, Minematsu, Yamada, Den, Utsuro, and Sagayama 2004) is a toolkit for developing anthropomorphic spoken dialogue agents, providing basic functions to achieve incremental speech recognition, mechanisms for lip synchronization, highly customizable speech synthesis and recognition, realistic face animation, and a virtual machine architecture enabling transparency in inter-modular communication. Four major requirements were defined for the toolkit:

1. *Natural spoken dialogue* - The toolkit must provide basic functions to represent human-interaction phenomena such as acknowledgments and head movements. Also, dialogue speed, quality, and balance must be taken into account, in order to prevent the user from feeling strange, incapacitating natural communication.

2. *High customizability* - The toolkit must be designed for multi-purpose, allowing easy voice, face and task customization.

3. *Modular architecture* - The functional units must be modularized in order to enable parallel functioning and independent development.

4. *Free open-source* - The toolkit and its source code must be released freely, in order no enhance contributions from developers and researchers.

In order to fulfill these requirements, the authors came up with the architecture shown on figure 2.2, which introduces three modular functional units for speech recognition, speech synthesis and facial animation, as well as a task manager which defines the actions to execute in the current context and an agent manager which aggregates all the previous and provides an API for the users.

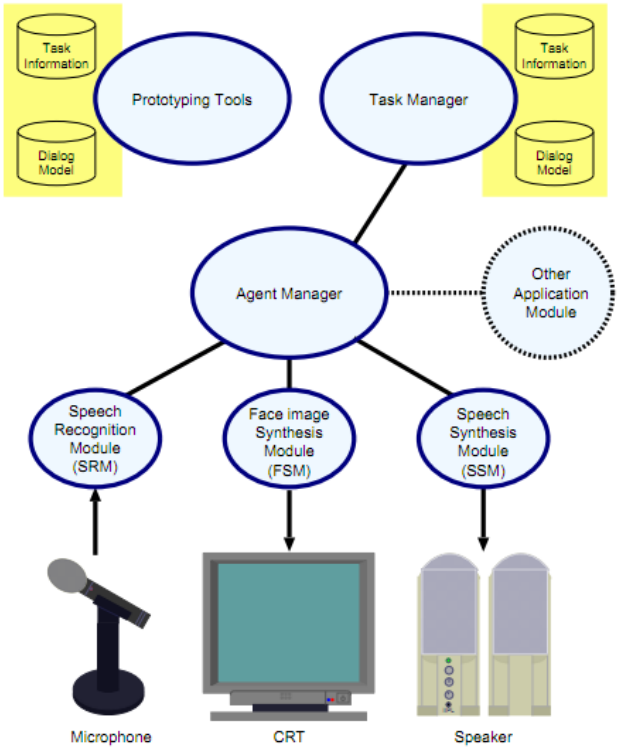

Figure 2.2: Galatea Architecture (Kawamoto, Shimodaira, Nitta, Nishimoto, Nakamura, Itou, Morishima, Yotsukura, Kai, Lee, Yamashita, Kobayashi, Tokuda, Hirose, Minematsu, Yamada, Den, Utsuro, and Sagayama 2004)

Like TRIPS, Galatea has a great focus on modularity, taking it even further by using a virtual machine model with a common communication interface, so that the agent manager can easily connect with every model, functioning has a hub for the whole system.

This toolkit's characteristics can help us in our project; however, if we want to use it, we will have to extend it, so that dialogue can be used as a learning tool. Also, as we want our system to have a physical front-end which may not be anthropomorphic, we will have to adapt communication to bodies that may not be able to provide facial animation.

2.2.3 Olympus

Like the two previous systems, Olympus (Bohus, Raux, Harris, Eskenazi, and Rudnicky 2007) is a framework for research in conversational interfaces which has been used to develop and deploy systems covering different domains and interaction types. It was developed as an attempt to bridge the gap between industry and academia, supporting and fostering research, as the first has the funds and the second has the freedom to research questions with higher risks. In order to achieve that, the authors concluded that the framework should have five essential characteristics:

1. *Open* - All the framework's source code must be available for free, in order to allow modification toward different ends, enhancing the system.

2. *Transparent* - The framework must provide data visualization and analysis tools and every component shall provide detailed information about their inner state.

3. *Flexible* - The framework must be able to accommodate multiple kinds of applications and purposes as well as easy integration of new technologies.

4. *Modular* - Specific functions must be encapsulated, in order to promote reusability and reduce development effort.

5. *Scalable* -The framework's ability to scale from simple to complex environments must not be compromised for the sake of flexibility and transparency.

In order to keep these characteristics, the system was developed according to the architecture shown on figure 2.3, which reveals a pipeline architecture starting with the capture of the audio signal corresponding to a utterance, which is then analyzed by the speech recognition module in order to generate a recognition hypothesis from which the parser can extract relevant information. This information is then annotated with a confidence score and passed to the dialog manager which integrates it in the current context and decides the next action to be taken. This action is analyzed by the language generator in order to translate it into natural language and then passed to the speech synthesizer to be rendered as audio. Figure 2.3 displays concrete implementations to these components; however, they can be replaced by any others that satisfy the specifications.

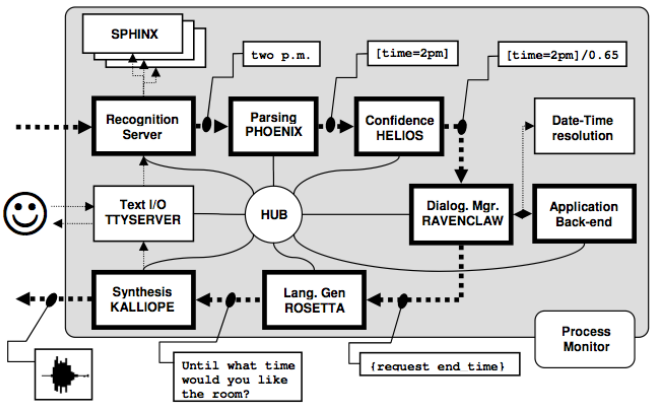

Figure 2.3: Olympus Architecture (Bohus, Raux, Harris, Eskenazi, and Rudnicky 2007)

The pipeline architecture may seem straightforward; however, in order to keep independence between the different components, all the communication is centralized in a hub, which prevents direct communication and thus, dependence, between the components, but has the downside of requiring much more specifications.

This framework can be of great usefulness for the dialogue part of our project, as it has all the characteristics we need: modularity, so we do not need to be dependent of a concrete component; transparency, in order to precisely evaluate the components performance; and flexibility and scalability, enabling multiple scenarios with different complexities.

2.2.4 DIGA - DIaloGue Assistance

DIGA (Martins 2008) is the in-house dialogue system, which was initially built for information retrieval through speech but afterwards evolved into a framework for other dialogue applications. It is based on TRIPS, which leads to the very similar architecture shown on figure 2.4. The audio input goes through a speech recognizer, which transforms it into text that can be parsed and transformed into speech acts that the interpretation manager can interpret according to the discourse context and the existing domain information, while also updating them. The resulting discourse obligations are sent to the behavioral agent which updates and executes the plan accordingly, by calling services from the task manager and sending dialogue obligations to the generation manager. The generation manager generates the utterance's structure and sends it to the surface generator which translates it into natural language that can be synthesized into speech.

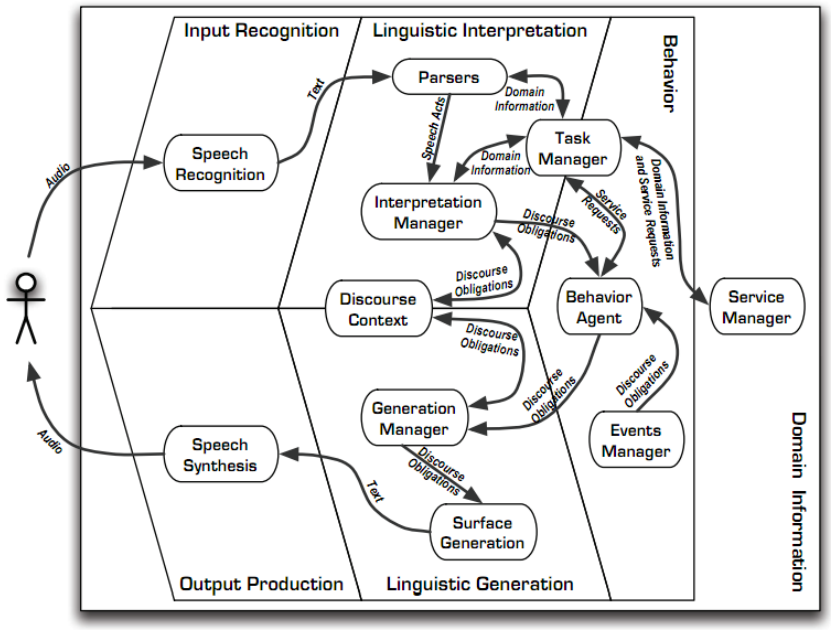

Figure 2.4: DIGA Architecture (Martins, Pardal, Franqueira, Arez, and Mamede 2008)

Although this is the in-house dialogue system, it may not be the most appropriate for our project, as it was developed for the portuguese language; however, that is something to be discussed along the development of our system.

2.3 Intelligent Agents

The system we want to develop is an embodied conversational agent, able to learn from dialogue and, thus, we must find the best ways to support a natural behavior and obtain knowledge from the environment which can help the system with future tasks.

Although some of the systems described in the previous section can be considered agents, they were separated from the following as they are frameworks built specifically for dialogue,

while the following are, respectively, a generic architecture and its implementation for intelligent agents and an embodied conversational agent which is similar to a part of the one we want to develop.

2.3.1 FAtiMA - FearNot Affective Mind Architecture

FAtiMA (Dias, Mascarenhas, and Paiva 2011) is an agent architecture with planning capabilities which uses emotions and personality to influence the agent's behavior. It was designed for the development of agents in the FearNot! Project (Paiva, Dias, Sobral, Aylett, Woods, Hall, and Zoll 2005) but was used in many others, such as the LIREC[1] Project, which led to several extensions of the initial architecture, making it confusing and difficult to use. For this reason, a modular version of the architecture was created, separating functionalities and processes into independent components, allowing the use of lighter versions of FAtiMA and simplifying extension. This modular architecture is composed of a core layer to which components can be added in order to provide additional functionality.

2.3.1.1 FAtiMA Core

The core layer is a template describing the general functioning of the architecture with generic functions for which the added components can provide specific implementations. Figure 2.5 shows the basic functionalities for an emotional agent architecture and the relations between them. The agent receives perceptions from the environment, which update its memory and trigger the appraisal process. This process' result is stored in the affective state of the agent, which influences the action selection process and thus, how the agent acts in the environment.

The appraisal process is probably the most important part of an emotional agent; however, we will not get deep into it as we are not planning to introduce emotions in our agent, at least for now. Once again from figure 2.5, we can see that the appraisal process is divided into two different processes. The first one, appraisal derivation, is responsible for evaluating the relevance of the perceptions for the agent's emotional state and determining a set of appraisal variables (e.g. desirability and praiseworthiness). The second one, affect derivation, is responsible for generating affective states from the variables determined by the appraisal derivation process, according to an appraisal theory.

2.3.1.2 FAtiMA Components

As previously mentioned, FAtiMA Core is just a template, thus, an agent that only has core will not do anything. This is where the components come in, by being added to the core, in order to achieve the desired functionality. Many of them are already implemented and, thanks to the modular architecture, new ones can be easily added. The following components are included in the FAtiMA project:

- *Advanced Memory* - contains functionalities such as memory retrieval and forgetting.
- *Culture* - introduces culture dependent behavior through the use of rituals and cultural dimensions.

[1] http://lirec.eu/

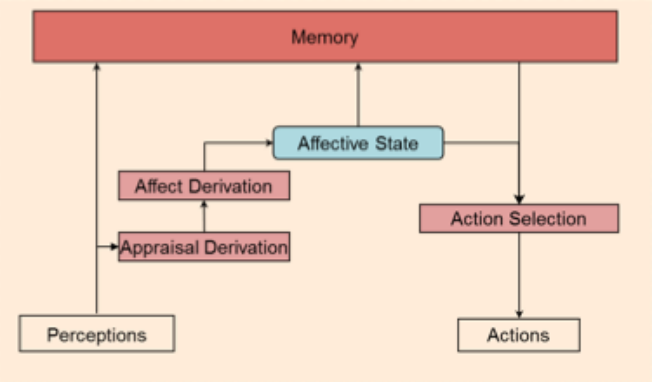

Figure 2.5: FAtiMA Core Architecture (Dias, Mascarenhas, and Paiva 2011)

- *Deliberative Component* - enables deliberative behavior by adding BDI and planning capabilities to the agent.
- *Emotional Intelligence* - allows the agent to reason about and plan based on emotions.
- *Empathy* - a model that allows the agent to perform emphatic appraisal, triggered by other agents' emotions.
- *Language Server* - an engine used for natural language generation.
- *Motivational System* - uses PSI needs (e.g. energy, integrity and affiliation) to activate goals.
- *OCC Affect Derivation* - generates emotions according to the OCC theory of emotions (Ortony, Clore, and Collins 1988).
- *Reactive Component* - implements reactive emotion rules and their relation to actions.
- *Social Relations* - a model of social relations between agents (e.g. respect and friendship).
- *Theory of Mind* - allows the agent to have models of the other agents' minds and use that knowledge to achieve its own goals.

2.3.1.3 Conclusions

FAtiMA is an emotion-oriented architecture and, at this stage, we do not want to focus on emotions. However, FAtiMA is still an intelligent agent architecture, thus parts of it and its modular concept can be applied in our project at this stage, leaving an open door for future inclusion of emotions.

2.3.2 Greta

Greta[2] (Pelachaud, Carofiglio, Carolis, de Rosis, and Poggi 2002) is a conversational agent embodied in a 3D model of a woman, which is able to communicate using a wide range of verbal

[2] http://perso.telecom-paristech.fr/~pelachau/Greta/

and nonverbal behaviors. It can simultaneously talk, show facial expressions, gesture, gaze, and move her head. These features are very important to enable a believable conversation, as body language has an unquestionable role on human communication. Also, it has a personality and a social role, allowing it to show or refrain from showing its emotions according to the context.

Greta was designed to engage in information giving dialogues in the form of question / answer, such as giving information to patients as a medical assistant or explaining parts of a story as a storyteller. In order to achieve this kind of dialogue, the system follows the architecture presented in figure 2.6.

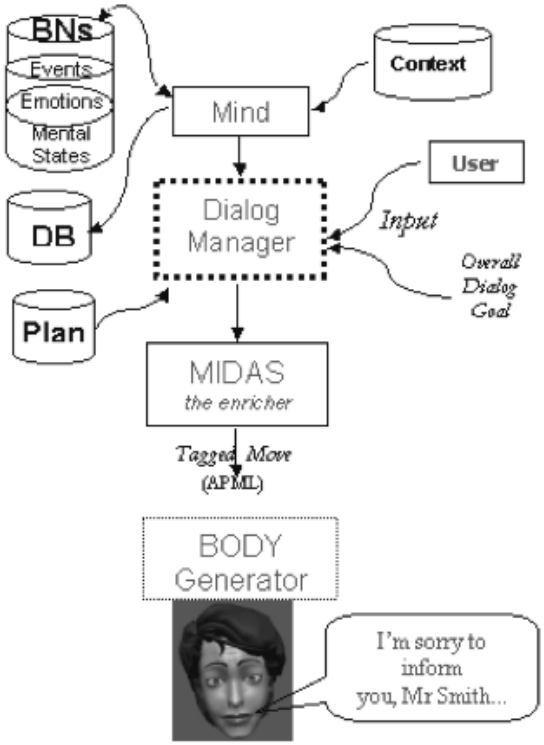

Figure 2.6: Greta Architecture (Pelachaud, Carofiglio, Carolis, de Rosis, and Poggi 2002)

When the dialogue starts, the goal in that particular domain is set and passed to the dialogue manager, enabling the creation of a discourse plan according to the agent's mind, which represents how the agent will try to achieve the communicative goal during the conversation. Once the dialogue manager selects the output, it asks the mind if there are any affective states to be activated and their intensities. In the next step, the output is enriched by the Midas module with the addition of tags indicating the synchronism between the verbal stream and other communicative functions. Finally, the enriched dialogue move is passed to the body generator, which interprets and renders it, producing the corresponding behavior.

2.3.2.1 Mind

The mind is responsible for deciding which affective states are active and their intensities, as well as if and how the felt emotions shall be displayed in the given context and how they affect the agent's goals.

The agent's mental state is represented as a dynamic belief network, which is updated at every dialogue turn. This network can have three different types of nodes, representing beliefs, goals, and goal-achievements. The agent's personality is seen in terms of the importance given to achieving different kinds of goals. Emotions are triggered by the belief of being close or distant from those goals and tend to decay over time, according to the agent's personality.

2.3.2.2 Dialogue Manager

The dialogue manager is built on top of a toolkit (Larsson and Traum 1999) which provides a computational model of dialogues. The dialogue flow is controlled through the iteration of the following steps:

1. Select an appropriate discourse plan for the given goal and generate the first dialogue move, that is, the first utterance, setting the goal as the main topic of the conversation.
2. Wait for questions on the topics under discussion.
3. Translate the question into a symbolic communicative act.
4. Select which sub-plans to execute and generate the dialogue move.
5. Go to step 2.

The output is a discourse plan represented as an XML-tree. Also, a representation of the conversational partner's move is sent to the mind, in order to update the mental state and associate emotions with the communicative acts.

2.3.2.3 Midas

The Midas module translates the symbolic representation of a dialogue move into a behavior specification by enriching it with the addition of tags representing communicative functions, which can be instantiated by the body generator.

2.3.2.4 Body Generator

The body generator module interprets the enriched dialogue move and chooses the actions to be made on each communicative channel, according to the tags.

2.3.2.5 Body Model

The body used is composed of an animated 3D face model, which shows expresses the nonverbal components of the dialogue move, and a speech synthesizer which deals with the verbal components.

2.3.2.6 Conclusions

Greta is a very good example of a conversational agent and some parts of its architecture can be adapted to our project, especially the dialogue manager. However, as said in the previous section, the emotional part is not the main focus of our project and, thus, our mind and the actions triggered will be very different. Also, similarly to the Olympus framework – section 2.2.3 – Greta does not include learning components, so, the dialogue model must be adapted in order to generate moves that enable learning. Furthermore, our physical bodies differ from Greta's 3D model and, thus, the system must be adapted to them.

2.4 Knowledge Representation and Reasoning

As we want our agent to learn and be able to talk about what it learned, knowledge plays a major role in our system so, we must find ways to store that knowledge and reason about it. Ontologies are the most used forms of knowledge representation for applications like ours and there are engines and reasoners that facilitate the retrieval and addition of knowledge from and to those ontologies.

2.4.1 Ontologies

By performing an online search for publicly available ontologies, it was possible to find more than 60 different ones, covering many different subjects, from colors to genetics. It is important to note that this was a search for ontologies in general and, therefore, there is a high probability that many more can be found if the search is narrowed to specific subjects. Most of the ontologies found are in the OWL – Web Ontology Language[3] – format, although a few are in simple RDF – Resource Description Framework[4].

These ontologies can be very useful in the project, as there might be cases where we do not want the agent's knowledge on a subject to start from scratch, thus using the knowledge present in ontologies as a baseline which can be reasoned about and improved through learning.

2.4.2 Jena

Jena[5] is a framework for building Semantic Web applications which provides a programmatic environment for RDF, RDFS – RDF Schema[6] – and OWL and includes a rule-based inference engine using SPARQL – SPARQL Protocol and RDF Query Language[7]. Overall, the framework features:

- A RDF API
- Support for reading and writing RDF in RDF/XML, N3 and N-Triples

[3] http://www.w3.org/2004/OWL/
[4] http://www.w3.org/RDF/
[5] http://openjena.org
[6] http://www.w3.org/TR/rdf-schema/
[7] http://www.w3.org/TR/rdf-sparql-query/

- An OWL API
- Support for in-memory and persistent storage
- A SPARQL query engine
- Many internal reasoners
- Support for external reasoner setup

This framework can be very useful for our project, especially because of its inference engine which allows us to obtain knowledge from ontologies and update it through simple queries.

2.4.3 FaCT++

FaCT++ (Tsarkov and Horrocks 2006) is a description logic reasoner designed as a platform for experimenting with tableaux and optimization algorithms. It includes most of the standard optimization techniques, such as absorption and model merging, but also new ones, like a ToDo list architecture. This architecture is better suited for complex tableaux algorithms, such as those used to reason with OWL ontologies, and enlarges the range of possible heuristic optimizations.

The first stage a knowledge base goes through when being reasoned using FaCT++ is preprocessing, in order to normalize and transform it into an internal representation. After that, the knowledge base is prepared for classification, that is, the computation and caching of the subsumption partial ordering of named concepts. In order to perform this classification, a knowledge base satisfiability checker is used. This is the core component of the system, which decides what are the subsumption problems raised by a given pair of concepts.

All of the previous steps were highly optimized, in order to enable an useful ontology reasoning.

2.4.3.1 Preprocessing Optimizations

- *Lexical Normalization and Simplification*: A standard rewriting optimization which transforms all the concepts into a simplified normal form where negation, conjunction, universal restriction and at-most restriction are the only operators. This transformation is mainly used for early inconsistency detection; however, it can also simplify concepts and detect some inconsistencies itself. In FaCT++ this optimization is applied on the fly, during the parsing process.At the same time, the knowledge base is also simplified through the elimination of redundant constants, expressions, and subsumers.
- *Absorption*: Another widely used rewriting optimization which tries to eliminate General Concept Inclusion axioms, as they highly reduce the performance of tableau based reasoning. In FaCT++, these axioms are eliminated through absorption into concept definition axioms or role domain axioms.
- *Told Cycle Elimination*: Definitional cycles can lead to several problems, especially when using algorithms that exploit the told subsumer hierarchy so, they must be eliminated, which is a relatively easy process consisting of an axiom refactorization.

- *Synonym Replacement*: If all the synonyms are converted into one representation, simplification possibilities are increased and, thus, so is the possibility of early inconsistency detection. FaCT++ uses a canonical name for every set of synonyms in all the knowledge base's concept expressions.

FaCT++ preprocessing starts by translating all concepts into the simplified normal form, which is kept by every future transformation. After that come simplification and absorption. Finally, cycle and synonym elimination steps are repeated until the knowledge base stabilizes.

2.4.3.2 Satisfiability Checking Optimizations

- *Dependency-Directed Backtracking*: A crucial optimization which labels each concept in a completion tree with the correspondent dependency set, so it can backtrack to a previous branching point to try to eliminate inconsistencies.

- *Boolean Constant Propagation*: Another widely used optimization which is applied to formulas, in order to simplify them and reduce the number of nondeterministic rule applications.

- *Semantic Branching*: A rewriting optimization to be used together with the previous one which transforms disjunctions in way that if a part of it leads to inconsistency then it will not be added to the node by another expansion.

- *Ordering Heuristics*: Changing the order in which nondeterministic expansions are explored highly influences performance. Thus, the use of a good heuristic is very important for the system. As there is no master heuristic, FaCT++ selects which heuristics will be used by analyzing the knowledge base's structure.

2.4.3.3 Classification Optimizations

- *Definitional Ordering*: A well known technique which optimizes the order in which the knowledge base is computed according to its axioms' syntactic structure.

- *Told Subsumers and Disjoints*: The axioms' structure can also be used to avoid expensive subsumption tests, through the computation of obvious told subsumers and disjoints.

- *Model Merging*: A widely used technique which uses cached partial models to cheaply check for non-subsumption.

- *Completely Defined Concepts*: A new technique used in FaCT++ which identifies a subset of concepts that have all of their subsumption relationships completely defined. This subset allows taxonomy computation without subsumption tests.

- *Clustering*: A technique which creates new concepts, equivalent to the union of a group of sibling concepts and inserts them between the group and the parent, producing a deeper and more uniform structure.

2.4.3.4 Conclusions

FaCT++ is an evolved reasoner which uses a large set of optimizations to improve performance. It can reason with OWL-DL ontologies using the DIG (Bechhofer 2003) interface and, thus, it reason with most of the ontologies found. Also, we need fast reasoning for our project and FaCT++ can provide that. However, it lacks support for instance reasoning, as it only deals with classes.

2.4.4 Pellet

Pellet (Sirin, Parsia, Grau, Kalyanpur, and Katz 2007) is a sound and complete OWL-DL reasoner with support for reasoning with individuals, instead of just classes. Pellet provides the standard set of Description Logic inference services:

- *Consistency Checking* - Ensures that an ontology has no contradictory facts.
- *Concept Satisfiability* - Checks if it is possible for a class to have any instances.
- *Classification* - Computes all the subclass relations between named classes, creating the hierarchy.
- *Realization* - Finds the most specific classes that an individual belongs to.

In this reasoner, all of these services are reduced to Consistency Checking. In addition to these, it provides services for entailment, conjunctive query answering, and other more advanced ones, such as services for dealing with multiple ontologies at the same time.

Pellet was designed to work with OWL right from the beginning, which led its architecture, shown in figure 2.7, to be highly dedicated to it, making the reasoner both functional and accessible. This architecture focuses on extension, by having a small core, composed by the tableaux reasoner, to which many different components can be attached, enabling multiple interfaces.

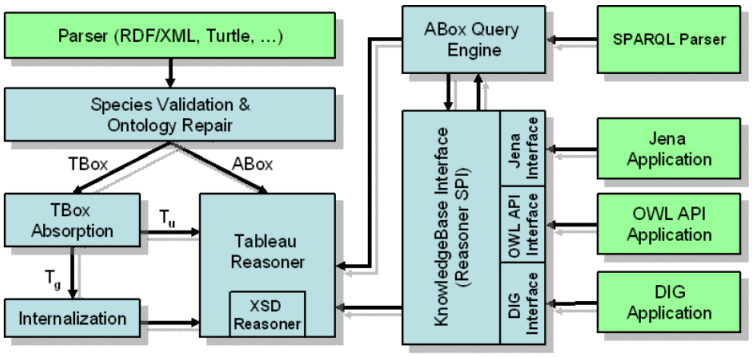

Figure 2.7: Pellet Architecture (Sirin, Parsia, Grau, Kalyanpur, and Katz 2007)

Although Pellet has slightly worse results than FaCT++ when reasoning with classes, it performs much better when reasoning with individuals, which can be very useful to our project if we want to refer specific objects, instead of just their classes.

2.5 Related Projects

Although our project is very complex and probably unique as a whole, there are many other projects, some already over but also some still in progress, which are similar to ours, at least partially. There has already been put a lot of effort into these projects and, thus, we should at least be aware of the research made on them so we know what to do on our project. The following are some of those projects.

2.5.1 Companions

Companions[8] (Mival and Benyon 2007) was a project funded by the European Union, involving a consortium of sixteen partners, from eight different countries. Its goal was creating embodied conversational agents which could help the eldest fight solitude and deal with the technological advancements, functioning as a multi-modal interface to the internet. This agents, called companions, should learn their owners' habits, needs and life memories, in order to assist them carrying out specific online tasks and store important information for relatives and friends.

The defined objectives for the project were to develop autonomous, persistent, affective and personal companions able to maintain a natural, believable and intuitive conversation, as well as interpret visual images and their contents.

Machine learning was on the core of the approach, especially for the learning of language input structure and dialogue frames, enabling a robust dialogue management capacity, regarding all the possible communication modalities.

One of the outcomes of the project was a scenario, following the architecture shown on figure 2.8 and based on a conversational system which establishes a relationship with the users and supports them emotionally. In this scenario the user and companion have conversations about daily life events and the latest is able to identify the key events and their emotional context. The conversations are based on turn-taking and the topic may be changed by both intervenients. The companion is able to appraise the user's affective state through the utterances' expressiveness and appropriately reply using different modalities, such as speech, gestures and facial expressions. Also, it uses this affective state appraisal to learn what is pleasant and unpleasant for the user.

Although this project has a strong emotional component, it is in many ways similar to ours, as it features embodied conversational agents which are able to learn which is exactly what we want. Therefore, we can learn from what has been done in this project and partially evaluate our results according to its ones.

2.5.2 LIREC - Living with Robots and Interactive Companions

LIREC[9] is an European project focused on the interaction between humans and physically embodied agents, called digital companions. The projects aims to the design and development of agents that can develop and read emotions and maintain long term relationships with humans, while acting on different platforms.

[8] http://www.companions-project.org/
[9] http://lirec.eu/

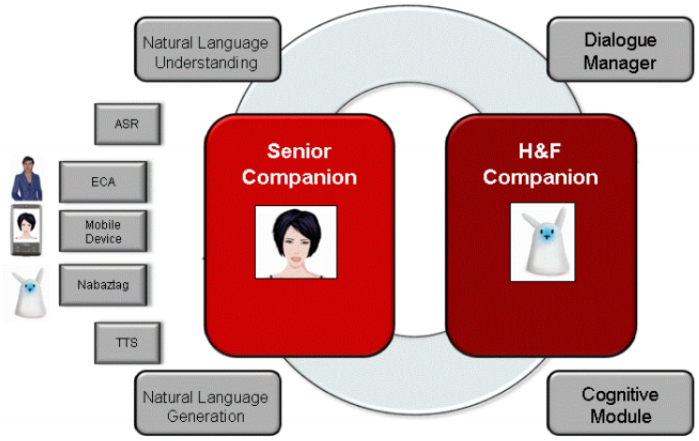

Figure 2.8: Companions Architecture (Zovato and Danieli 2008)

This project addresses many scientific and technical areas which were not given much attention before, such as:

- A theory of artificial long-term companions, including perception, memory, emotions, communication, and learning.
- A practical evaluation under real and adequate social settings to validate the theory.
- Ethological issues in developing these long term companion relationships.
- The effects of embodiment and body migration on long term relationships.
- The creation of a generic framework for viable long term companion development.

The developed digital companions follow the architecture presented on figure 2.9, which is a layered architecture with three levels. The bottom layer is the hardware level and contains the actual sensors and actuators while the middle layer is a software abstraction layer for the physical layer and contains the competences. The top layer is the agent's mind and is responsible for memory, reasoning and action selection. This top level is an implementation of the FAtiMA architecture presented on section 2.3.1.

As this project focus on emotions and long term relationships, it is not that similar to our project, at least at this stage. However, it features artificial agents interacting with humans and so there are intersection points between the LIREC project and ours and thus we can learn from what has already been done and be prepared if we decide to focus on emotions in the future.

2.5.3 CoSy - Cognitive Systems for Cognitive Assistants

CoSy[10] (Christensen, Kruijff, Sloman, and Wyatt 2010) was another European project with the goal of advancing the state of the art on cognitive systems through the investigation of requirements, design options and trade-offs for human-like, autonomous, integrated, physical

[10] http://www.cognitivesystems.org/

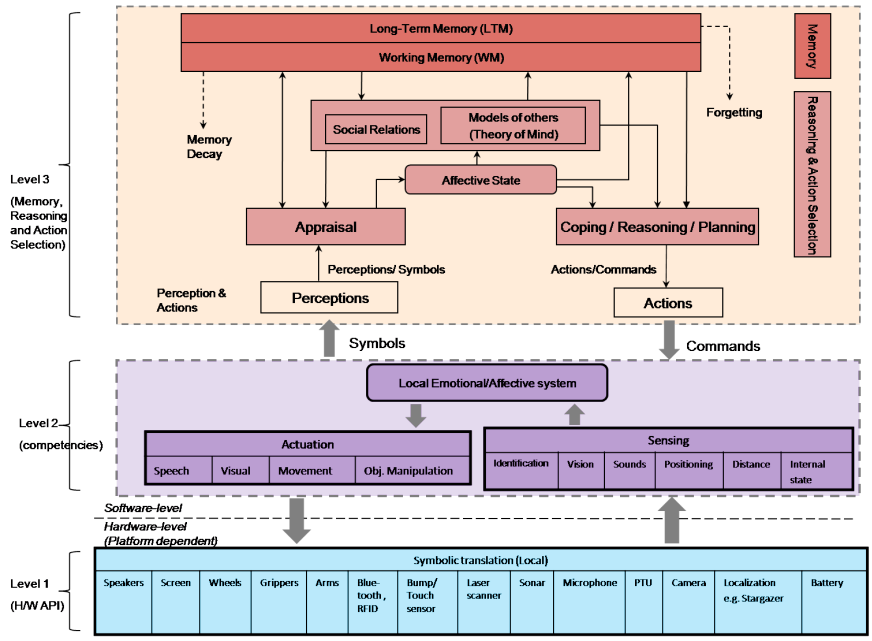

Figure 2.9: LIREC Architecture

systems. Requirements for architectures, representation, perception, learning, planning, reasoning, motivation, action and communication were investigated, in order to create such systems.

The creation of a highly competent robot, able to combine different capabilities in a coherent manner, like a very young human child is able to was defined as the most ambitious and distant goal of this research. A system like this would have a tremendous scientific importance as it would be a generic platform which could learn many different tasks and thus, developed in multiple directions, for multiple purposes.

The research in this project was scenario-based, enabling the study of different capabilities and requirements. Also, the produced architecture is composed of distributed subcomponents, called sub-architectures, each one with a working memory and a task manager. This subdivision enables the different scenarios, as competences can be added or removed according to necessity.

One of the developed scenarios was one where a robot was able to manipulate objects and talk about them with a human. This scenario requires many abilities from the robot, such as knowing how to reference those objects and how its actions affect the objects, detecting and recognizing actions the human performs and learning the effects of both its and externals actions on the objects. The system's architecture for this scenario is the one shown on figure 2.10, which features seven sub-architectures for visual information processing, communication, spatial reasoning, object manipulation, knowledge representation and reasoning, motivational and planning control, and information binding between all the previous ones.

This scenario is in some ways very similar to what we want in our project, as it features an agent capable of learning and talking about a subject. However, contrarily from what we want in our project, the learning focus differs from the conversational focus, as it talks about the objects but only learns the effects of actions performed on them, while we want our agents

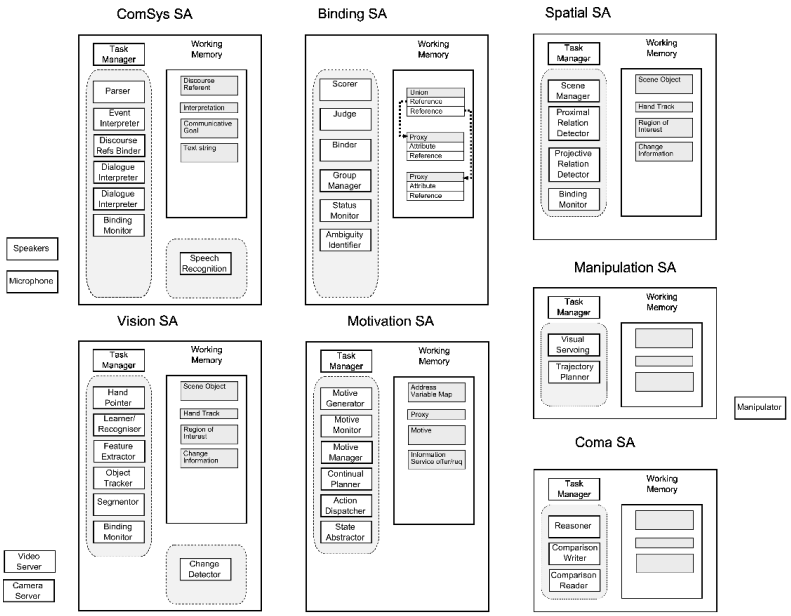

Figure 2.10: PlayMate Architecture (Christensen, Kruijff, Sloman, and Wyatt 2010)

to talk about what they learn.

This project made great advances in the research on cognitive systems and, thus, many of the techniques used in it should be used in ours and improved when possible and necessary, advancing some little steps towards the previously stated distant goal.

2.5.4 CogX - Cognitive Systems that Self-Understand and Self-Extend

CogX[11] is the project that followed up on the previous one and is devoted to developing cognitive systems able to function on open and challenging environments and deal with novelty and change. The project aims for the development of a self-understanding and self-extension theory, together with a practical deployment, in order to prove its value and efficiency.

A system like this must be able to model not only the environment but also its own understanding of it and the ways in which actions can modify that understanding. In order to achieve this, the system must identify gaps in its own understanding and try to fill them by planning how to obtain the necessary information and executing that plan. This process leads to the learning of new abilities and knowledge, which enables the system to perform future tasks more efficiently.

The systems developed in this project use the same architectural design of the ones from the CoSy project, as can be seen on figure 2.11, which shows the architecture for one of the systems used on the CogX project, called George, a robot able to learn simple visual concepts, such as colors and shapes. The architecture is very simple, only containing three sub-architectures for communication, vision processing and binding, that is, connection, between the previous two.

During the course of the project, research has already been made on situated dialogue (Kruijff, Janiček, Kruijff-Korbayová, Lison, Meena, and Zender 2009), so that the system could

[11] http://cogx.eu/

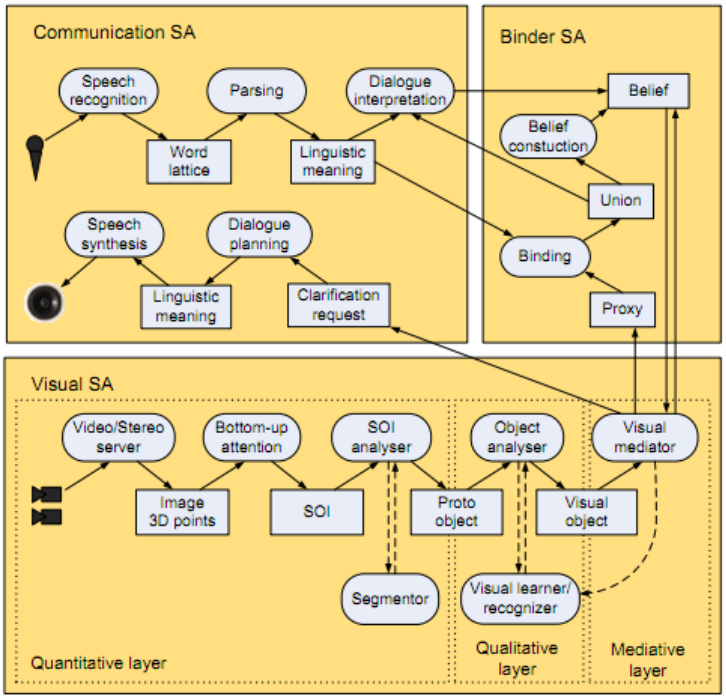

Figure 2.11: Architecture for one of the CogX's systems - George (Skocaj, Janicek, Kristan, Kruijff, Leonardis, Lison, Vrecko, and Zillich 2010)

use conversation as another tool to help it learn more about the environment. For this to be possible, human and robot must establish a common ground, allowing them to understand each other's needs and what are the subjects being addressed.

The research was focused on situated dialogue for continuous learning, where the system is mainly driven by its own curiosity instead of external factors. The system wanders the world, building its own categorizations and structures, however, when there are uncertainties, it turns to the human, looking for help and this is when situated dialogue can help the system extend itself. Nevertheless, the system has its own representation of the world, which can differ from the way the human sees it. Thus, for this kind of learning to be possible, transparency must be achieved. This is where the common ground comes in, so that the questions asked to the human clearly state what the system wants to achieve.

This kind of situated dialogue is almost exactly what we want in our project, except that we want the system to learn not only from its curiosity but also from external factors and experiences. Thus, the CogX project's results can be compared with ours for evaluation and we can take advantage of those results and build on them.

2.6 Summary

From section 2.2 we were able to notice that dialogue systems should be modular, with the modules as independent as possible, in order to enable easy replacement, extension and deployment for different scenarios. Furthermore, for the same reasons, the communication be-

tween the modules should be centralized.

Furthermore, on section 2.3, we presented two systems that are good references in the intelligent agents field. The first for an agent's intelligent behavior in general and the second for the development of an embodied conversational agent.

About knowledge representation, on section 2.4, we stated that ontologies are the most common way of representing an agent's knowledge nowadays and that there are multiple ones publicly available. In addition, we presented multiple ontology reasoners that are able to extract and update knowledge from the ontologies.

Finally, on section 2.5, we discovered that there are many projects similar to ours from which we can learn and with which we can compare our results. Some of them are only partially related, but one in particular, CogX, is very similar to ours, only differing on how and when the system is expected to learn.

3 Architecture

"I am the Architect. I created the Matrix. I've been waiting for you. You have many questions, and although the process has altered your consciousness, you remain irrevocably human. Ergo, some of my answers you will understand, and some of them you will not. Concordantly, while your first question may be the most pertinent, you may or may not realize it is also the most irrelevant."

– The Architect in The Matrix Reloaded

As we are developing a framework for the development of robotic agents for multiple scenarios, it must focus on extensibility, flexibility and adaptability, as the agents may have different needs and competences, as well as multiple physical bodies, for each scenario. The LIREC Project's (Section 2.5.2) architecture had a similar focus, thus, we took advantage of that architecture's strengths and came up with the functional, three-layer architecture presented on figure 3.1.

On the top layer is Brainiac, the mind, which stores and reasons about knowledge, defines objectives based on that knowledge and received perceptions, and plans actions to fulfill those objectives.

The middle layer aggregates the agent's competences and includes a competence manager, Mourinho, as well as multiple competence processors. As we focus on agents able to obtain knowledge through dialogue, we included a mandatory language processor, Jint, but many other processors may be added according to the scenarios and different physical bodies.

The bottom layer is the agent's body, which is physical most of the times and can have multiple and distinct capabilities.

We will now present each of the referred components in detail.

3.1 Brainiac: The Mind

When seen as a black box, Brainiac, the mind, is simply a component that receives perceptions and somehow generates actions that shall be executed. However, that how is its most important and complex part, as it involves perception processing, knowledge storage and reasoning, objective management and action selection.

As we are expecting the framework to be able to function in multiple scenarios, with different bodies, which may have different competences, and even deal with online body changes, we developed a model for the mind, the Knowledge-Objective-Competences Model, which performs action selection according the the current body's competences. Furthermore, it is prepared to deal with the dynamic knowledge updates inherent to a learning agent.

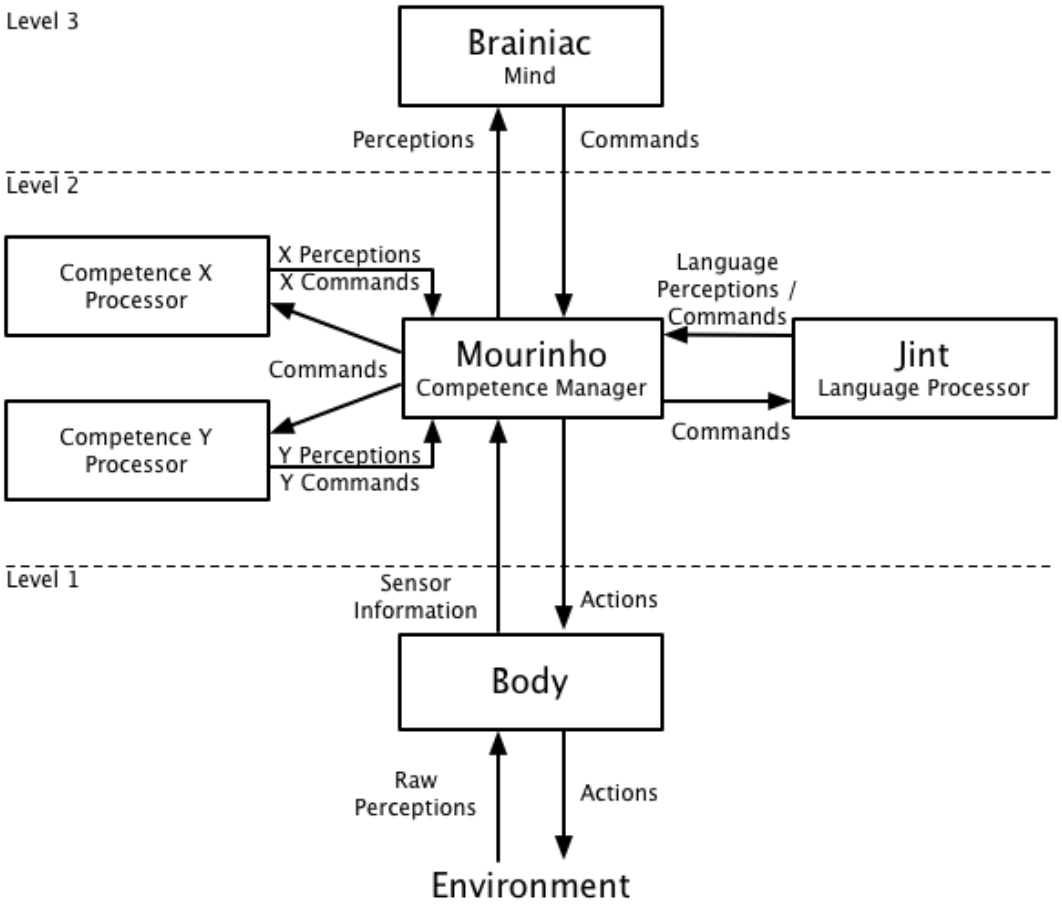

Figure 3.1: The system's architecture

3.1.1 The KNOC Model

The KNOC – Knowledge-Objectives-Competences – Model, is a mind model which, as the name indicates and can be seen on figure 3.2, has three main focuses – Knowledge, Objectives and Competences.

Knowledge management, the crucial task for a learning agent, is performed by resorting to a knowledge base which separates knowledge into two different categories – short term memory and long term memory – where the former represents the most recently received perceptions and the latter the knowledge obtained by the agent along time through experience, learning and reasoning.

As for the objectives, they are generated and updated by reasoning with both knowledge categories present in the knowledge base, a process which is able to update knowledge as a side effect.

However, reasoning is not always enough to fulfill an objective and that is where the competences become handy, as by knowing which are the competences of the current body the mind is able to generate a plan with a set of supposedly valid actions that may be executed by that body in order to fulfill an objective. Furthermore, it helps the mind deciding if an objective

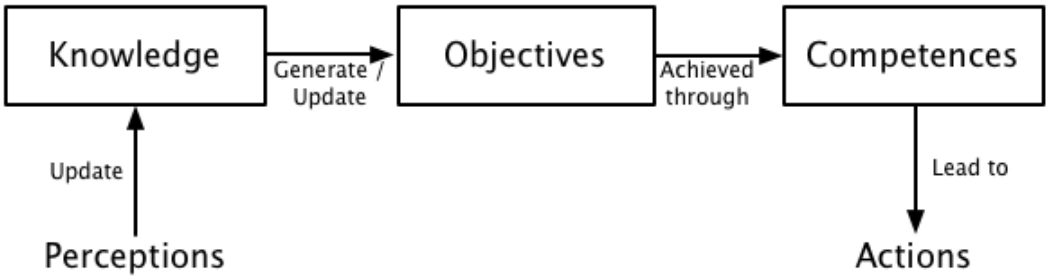

Figure 3.2: The KNOC Model

cannot be fulfilled with the current body and, thus, should be abandoned, at least until the body as changed.

Following figure 3.2, we may see the KNOC Model as a cycle for which an iteration is the following:

1. New perceptions are received.
2. Interesting perceptions are stored as short term memory while the others are discarded.
3. Objectives are generated and updated through reasoning with the existing knowledge.
4. If there are no objectives requiring physical action, the iteration ends here.
5. Otherwise, an action plan for each objective is generated according to the agent's competences.
6. If no plan could be generated for a given objective, it is put aside until there are competence changes.
7. The action plans are executed, hopefully leading to new perceptions and, thus, to a new iteration of the cycle.

So that Brainiac's functionality could follow the KNOC Model, we designed the manager approach presented in the next section.

3.1.2 The Manager Approach

We believe that Brainiac's functionality may be divided into four complementary high level tasks – perception processing, knowledge representation and reasoning, objective management, and action plan execution – and, thus, that an independent module should exist for each of these tasks, being responsible for its management.

As can be seen on figure 3.3, there are four managers – Perception Manager, Knowledge Manager, Objective Manager, and Execution Manager – responsible for each of the tasks, respectively. We believe that, together, these four managers are able to fulfill the requirements and function of the KNOC Model.

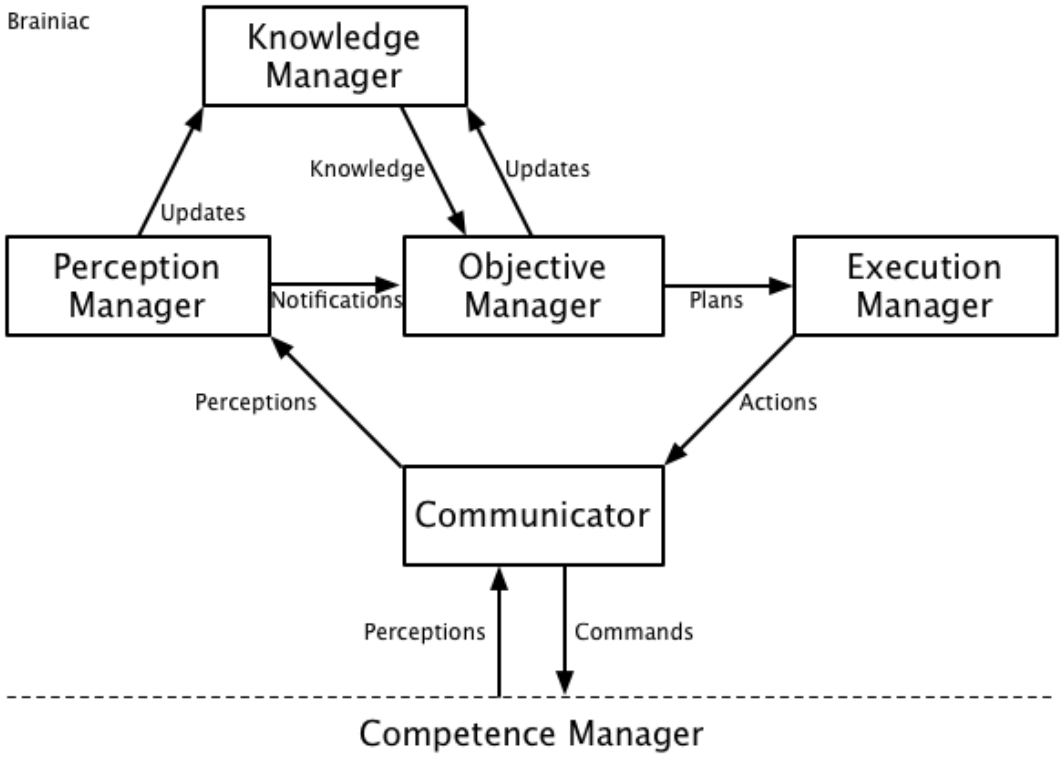

Figure 3.3: The Manager Approach for Brainiac

Furthermore, we also included a Communicator module that deals with communication with the competence manager, so that dependencies from specific communication interfaces and protocols are avoided in the management modules.

3.1.2.1 Perception Manager

The Perception Manager is responsible for receiving unprocessed perceptions from the Communicator module and processing them.

The perceptions, represented as predicates, are first of all checked for importance. This importance refers both to the validity of the perceptions and they're interest for the agent in the current state, that is, wether they may be able to influence the agent's knowledge and objectives. Non important perceptions are immediately discarded, while the others are sent to the Knowledge Manager as short term memory updates.

Furthermore, if important perceptions were received, the Perception Manager notifies the Objective Manager of the existence of that new and possibly useful knowledge.

3.1.2.2 Knowledge Manager

The Knowledge Manager functions as a knowledge base, as it is responsible for storing all the mind's knowledge. As previously stated when talking about the KNOC Model, knowledge is separated in two different categories – short term memory and long term memory – where the

former represents the most recently received perceptions and the latter the knowledge obtained by the agent along time through experience, learning and reasoning.

While short term memory is transient, as it is replaced when updates are received from the Perception Manager, long term memory is generally persistent, as it contains the knowledge obtained by the agent along time. The exception to the latter occurs when there are knowledge conflicts and, thus, part of that knowledge must be discarded or go through some kind of process for the elimination of those conflicts. It is important to notice that updates to long term memory and conflicts they may implicate, come from the Objective Manager, as part of its reasoning process.

We decided not to include a mandatory kind of knowledge representation in the framework, as according to the scenario and learning expectations, different kinds of representation might be preferred. However, at least for complex scenarios, we suggest the use of ontologies, as presented on section 2.4.1.

3.1.2.3 Objective Manager

The Objective Manager is responsible for managing the agent's desires and goals at each moment, as well as trying to fulfill them.

When notifications of new knowledge are received from the Perception Manager, the Objective Manager requests that knowledge from the Knowledge Manager and verifies if it can be used to generate new objectives or update the existing ones. New objective generation accounts for the expected function of the agent in the scenario it is actuating on.

When a new objective is created or an existing one is updated due to new short term memory knowledge, the Objective Manager reasons with the long term memory knowledge from the Knowledge Manager until the objective is accomplished or a state where physical action is required is reached. This process leads to updates to the Knowledge Manager's long term memory, as it represents the agent's capability of acquiring new knowledge through learning and reasoning.

When a state where physical action is reached, the Objective Manager generates an action plan to overcome that state, or even to fulfill the objective as a whole. In this planning process, actions are selected according to their capability to generate or enable the capture of relevant knowledge for the completion of the objective, and the competences of the current body, as differ from body to body and over time and limit the subset of possible actions. The generated action plans are then sent to the Execution Manager to be executed.

When the agent is not able to perform physical action to contribute for the completion of a given objective with the existing competences, that objective is put aside until changes in the agent's competences occur.

3.1.2.4 Execution Manager

The Execution Manager is responsible for storing and executing plans of actions along time. It receives the plans from the Objective Manager and sends the actions in chronological order to the Communicator module, which later sends them as commands to the competence manager.

The existence of a separate module to deal with action plan execution is useful, as it removes functionality from the Objective Manager, freeing it to perform its reasoning tasks.

Furthermore, with all the action plans being stored in the Execution Manager, it is possible to keep an action execution history, as well as avoiding the repetition of actions present in multiple plans that are chronologically close to each other.

3.1.2.5 Communicator

As previously stated, the Communicator module is responsible for communication with the competence manager, avoiding dependencies from specific communication interfaces and protocols in the management modules.

It receives the predicate perceptions from the competence manager and sends them to the Perception Manager to be processed. On the reverse way, it receives actions from the Execution Manager and sends them as commands to the competence manager, so that they can be dispatched to the corresponding competence processor.

3.2 Mourinho: The Competence Manager

The competence manager functions as a hub with which every other component communicates, in an attempt to keep the independence between the components. It is responsible for dispatching commands from the mind and sensor information from the body to the correspondent competence processors, as well as dispatching perceptions from the later to the mind and actions to the body. Furthermore, it is responsible for knowing the current body's competences and dealing with body changes, which is achieved through a body abstraction interface, which will be presented on section 3.4.1.

As can be seen on figure 3.4, Mourinho is divided in three modules – Communicator, Perception Dispatcher, and Command Dispatcher – responsible for communicating with the other components, dispatching perceptions, and dispatching commands, respectively.

3.2.1 Communicator

Similarly to what happens in Brainiac, the Communicator module is responsible for communicating with the other components, avoiding dependencies from specific communication interfaces and protocols in Mourinho's inner modules.

It receives the predicate commands from the mind and competence processors and sends them to the Command Dispatcher module to be dispatched. Later, it receives those commands together with their destination, as well as any sensor information necessary for their execution and sends them to the correspondent competence processors. However, some of the commands are mapped into actions and, thus, are sent to the body instead.

In the opposite way, it receives sensor information from the body and perceptions in the form of predicates from the multiple competence processors and sends them to the Perception Dispatcher module. Later, the perceptions are received again after any needed processing and dispatched to the mind.

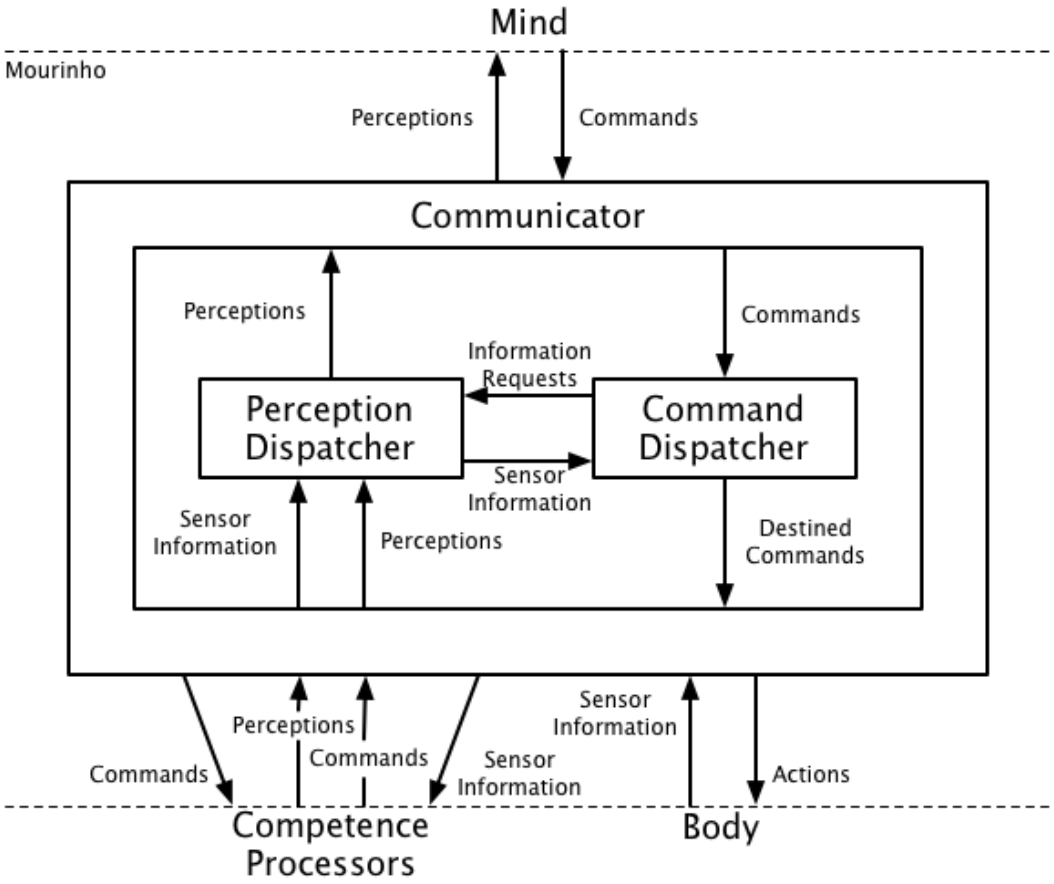

Figure 3.4: Mourinho: The Competence Manager

3.2.2 Command Dispatcher

The Command Dispatcher module is responsible for receiving commands from the Communicator module and executing four different tasks on them.

The Command Dispatcher's most important task is identifying the destination of each command according to its type. The destination may be either a competence processor or the agent's body.

Also, depending on the type of command, it may need to be paired with sensor information in order to be executed. This information is obtained from the Perception Dispatcher module and coupled with the previously existing command.

Furthermore, if a command's destination is the agent's body, it must be mapped into actions that the current body is able to execute. That is done by using the previously referred Body Abstraction Interface (Section 3.4.1).

Finally, the command's destination is appended to the command and it is dispatched to the Communicator module so it can reach its destination.

3.2.3 Perception Dispatcher

The Perception Dispatcher module's main responsibility is, as the name indicates, dispatching the perceptions received from the Communicator module to the mind, also through the Communicator module. However, it is also responsible for pre-processing those perceptions, identifying and discarding invalid ones.

Furthermore, the Perception Dispatcher also receives sensor information from the body, through the Communicator module. Although the great majority of this information is simply stored, waiting for future requests from the Command Dispatcher module, the Perception Dispatcher is also able to interpret part of it as perceptions, without requiring processing from any competence processor. This ability is normally used to deal with exception information, such as the one from crash sensors.

3.3 Jint: The Language Processor

Jint is a language processor, responsible for managing the speaking and listening competences of the agent. On the one hand it receives speech commands from the mind in the form of predicates and translates them into natural language, generating the corresponding audio signal. On the other hand it receives the audio signal from the body's sensors and recognizes speech in it, generating the corresponding textual form, which is later translated into predicates and sent to the mind in the form of language perceptions.

As can be seen on figure 3.5, Jint has three main modules – Translator, Speech Synthesizer and Speech Recognizer. The first one is present on both the previously referred tasks, as it is responsible for translating predicates into textual natural language and vice-versa. Each of the other two modules is responsible for the other important part of the corresponding task, with the speech synthesizer being able to generate the audio signal from the corresponding textual sentence and the speech recognizer being able to do the reverse process.

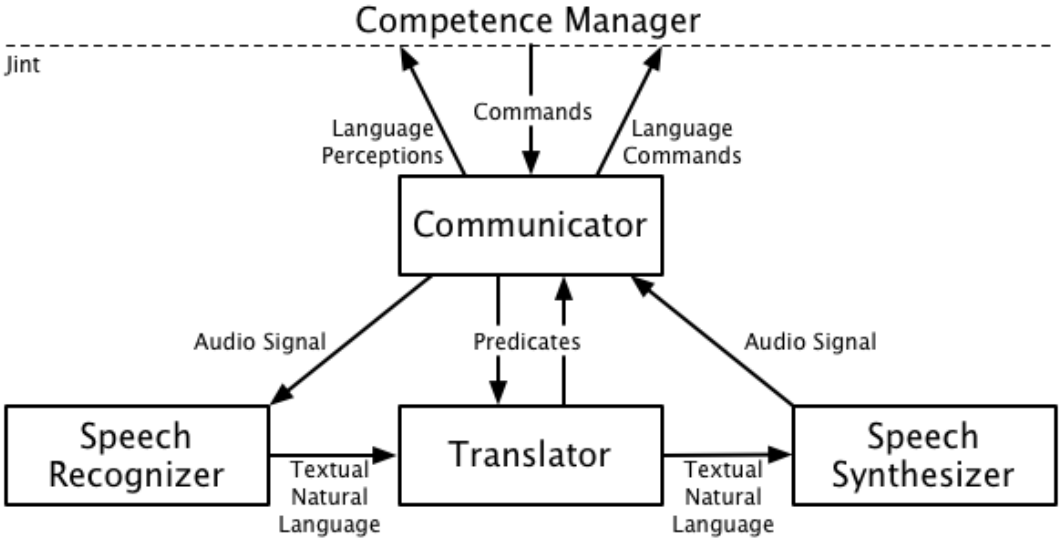

Figure 3.5: Jint: The Language Processor

Furthermore, Jint also has a Communicator module responsible for the abstraction of the communication interface and protocol, reducing the number of dependencies in the other modules.

3.3.1 Translator

The Translator module is responsible for translating textual natural language into predicates understandable by the mind and vice-versa.

On the one hand, it receives the textual sentences captured by the Speech Recognizer module and interprets them using a Language Interpreter of the agent's language (e.g., English Interpreter). This Language Interpreter must be able to capture the relevant information in each sentence, transforming it into predicates stating its linguistic function (e.g., Question, Statement) and, in the cases where it applies, any objects and properties it refers to. These predicates are then sent to the Communicator module, so that they can be forwarded to the competence manager and later to the mind, as perceptions.

On the other hand, the Translator module receives predicates from the Communicator module which were sent from the competence manager as commands. The Translator module translates these predicates into textual natural language using a Language Generator of the agent's language (e.g., English Generator). This Language Generator is able to map the predicates into natural language according to their function and, in the cases where it applies, objects and properties it refers to and the certainty associated with it. The generated textual sentence is then sent to the Speech Synthesizer module for its audio signal to be generated.

3.3.2 Speech Synthesizer

The Speech Synthesizer module is responsible for generating the audio signal corresponding to the textual natural language sentences received from the Translator module. That audio signal is then sent to the Communicator module, in order to be forwarded to the competence manager and later to the body, so it can be reproduced.

We will not present any speech synthesizer in detail as there are multiple available which may be used. Examples of these are MARY TTS[1] (Schröder and Trouvain 2003) and FreeTTS[2].

3.3.3 Speech Recognizer

The Speech Recognizer module is responsible for capturing textual natural language in the audio signal received from the body through the Communicator module. The captured sentences are then sent to the Translator module, in order to be translated into predicates that, as previously stated, will later be forwarded to the mind as perceptions, through the competence manager.

For the same reasons we did not present any speech synthesizer, we will not present any speech recognizer. Some of the existing speech recognizers are CMU Sphinx[3] (Lee, Hon, and

[1] http://mary.dfki.de/
[2] http://freetts.sourceforge.net/
[3] http://cmusphinx.sourceforge.net/

Reddy 1990) and Julius[4] (Akinobu Lee and Shikano 2001).

3.3.4 Communicator

Similarly to what happens in the other components, the Communicator module exists to avoid dependencies from specific communication interfaces and protocols in the other modules.

In the Jint's case, the Communicator module is responsible for the communication with the competence manager, receiving commands from it and extracting the coupled predicates, which are sent to the Translator module, or audio signal, which is sent to the Speech Synthesizer module.

On the other way, it receives the audio signal from the Speech Recognizer module and sends it to the competence manager coupled with a command for its reproduction. Also, it receives the predicates from the Translator module and sends them to the competence manager as perceptions, which are later dispatched to the mind.

3.4 Body

We want our framework to easily function with multiple, different bodies and even to be able to deal with body changes in runtime, However, these different bodies probably have different competences and most certainly have different interfaces. Given this, we had to come up with a way of dealing with these different bodies without compromising the system's ability to function.

The solution we found was the creation of a Body Abstraction Interface, based on the competences that a given body might have, which functions as an adapter for every body that might be used with our system.

3.4.1 Body Abstraction Interface

The Body Abstraction Interface functions, in part, as the Communicator modules presented for all the other components, as it deals with the communication with the physical body without introducing dependencies from specific communication interfaces and protocols, which is needed as they differ from body to body. However, the Body Abstraction Interface is more than that as it is able to activate only the functions that the current body is able to perform, based on competences. Furthermore, the execution of those functions is abstracted, enabling a silent transition between bodies with the same competences, even when the methodology used to achieve a certain end differs.

About competences, as they are responsible for enabling or disabling certain functions, they can help the mind decide the best way to achieve a certain objective according to the existing body. This process was previously presented on section 3.1.1. The competences of the current body are managed by the competence manager, which informs the mind of any changes, so that action planning may be updated accordingly.

[4] http://julius.sourceforge.jp/

We have multiple bodies available, including three WowWee Rovios[5], a WowWee RS Media[6], a Meccano Spykee[7], a Parrot AR.Drone[8] and a Magabot[9], which have distinct competences. Although these robots have a large amount of different sensors and actuators, which may lead to multiple different competences, for a first version of the Body Abstraction Interface we only included a subset of the possible competences and respective functions as they were the ones we considered most relevant for our framework. However, new competences may easily be added, without the need for messing with instances of the interface developed for bodies where those competences are not relevant.

The chosen competences and corresponding subset of functions were the following:

Moving: A competence that represents the body's ability to change its position.

changeSpeed(speed): A function for changing the movement speed.

rotate(direction, speed): A function to rotate in a given direction, at a given speed.

move(distance, direction, speed): A function to move a given distance, in a given direction, at a given speed.

Seeing: A competence that represents the body's ability to capture visual information from the environment.

captureFrame(): A function that captures a single frame from the camera(s).

captureVideo(time): A function that captures video input from the camera(s) during a given time period.

Hearing: A competence that represents the body's ability to capture audio information from the environment.

captureAudio([time]): A function that captures audio input from the microphone(s) during a given time period or until silence is detected.

Speaking: A competence that represents the body's ability reproduce an audio signal.

playAudio(audio): A function that reproduces a given audio signal.

Displaying: A competence that represents the body's ability to display visual information.

displayFrame(frame): A function that displays a single image.

displayVideo(video): A function that displays a video clip.

displayText(text): A function that displays textual information.

It is important to notice that, although a given body has a certain competence, it is possible that it is not able to execute some of the functions from the subset enabled by that competence. However, when comparing with the case where there is no notion of competence, these cases

[5] http://www.wowwee.com/en/products/tech/telepresence/rovio/rovio
[6] http://www.wowwee.com/en/products/toys/robots/robotics/robosapiens/rs-media
[7] http://www.spykeeworld.com/
[8] http://ardrone.parrot.com/
[9] http://magabot.cc/

occur with much lesser frequency and, thus, may be treated as exception cases instead of normal ones. Information from these exception cases can be used to complement the competence based approach, leading to a better action planning from the mind.

Furthermore, it is important to refer that although our framework accepts bodies with different competences, or with no competences at all, there are some competences that are mandatory for the agent to be able to achieve its purpose of learning through dialogue. These competences, Hearing and Speaking, must exist, as they play a very important role in dialogue. Also, it is recommended that the used bodies have at least another competence that enables them to obtain information from the environment (e.g., Seeing), for the possibility of coupling linguistic information with known physical properties.

3.5 Communication

From the previous sections we are able to conclude that each component has its independent module for communication, the Communicator module. These modules were introduced for the sake of flexibility and portability, as they remove dependencies from communication interfaces and protocols from each component's main modules. This leads to a situation where any communication interface or protocol may be used for the communication between the multiple components, as long as it can be used to perform the role of each Communicator module.

Although any communication interface or protocol may be used, we suggest the use of ones able to perform event based, asynchronous communication as, although synchronous communication is much simpler, it leads to a severe negative impact to the system's performance.

3.6 Configuration

As this is a framework for development, it must allow quick and simple changes to the system's configuration, in order to simply testing and the comparison of multiple solutions for a given problem. This is useful for calibrating the running parameters used for each component of the framework, as well as defining which external systems shall be used at a given time (e.g., speech recognizer and synthesizer).

Furthermore, the framework must provide a simple way of defining which actions and perceptions might occur in a given scenario, so that transition between scenarios is simplified.

In order to achieve this, we opted for the usage of configuration files, as they can be easily edited by the users of the framework. Also, we decided that each component must have a default configuration and that other configurations might be loaded by the users, from different locations, so that there is always a backup configuration present.

3.7 Summary

In this chapter we presented the framework's architecture, a three-layer, functional architecture, focused on extensibility, flexibility and adaptability.

On the top layer is Brainiac, the mind, responsible for storing and reasoning about knowledge, generating objectives, and generating plans to achieve those objectives, accordingly to the agent's competences. Brainiac performs these tasks according to the KNOC Model.

On the middle layer is Mourinho, the competence manager, which functions as a hub with which every other module communicates and, thus, is responsible for dispatching messages between the mind, body and competence processors. These competence processors, like Jint, the language processor, are also present on the middle layer and each one performs tasks related to the competence it is responsible for.

On the bottom layer is the agent's body, which is physical most of the times and can have multiple and distinct competences and change over time. For the framework to function with different physical bodies, a Body Abstraction Interface based on the body's competences is used, enabling only a subset of possible functions.

Furthermore, for the sake of flexibility and portability, each component has an independent communication module, as well as the possibility for multiple configurations, through configuration files.

4 Deployment

"When I left you, I was but the learner. Now I am the master."
– Darth Vader in Star Wars Episode IV: A New Hope

In order to test the value and validity of the theoretical framework presented on the previous chapter, we defined two test scenarios involving learning through dialogue. The first scenario is about a robot able to learn colors through dialogue with a human teacher, while the second is about a robot that meets people and is able to remember their names. Although these are relatively simple scenarios, with small, closed domains, they are both focused on learning through dialogue and, thus, are valid test cases for the framework's ability to deal with such kind of learning. Furthermore, they have different learning focus and domains and, thus, are also valid test cases for the framework's extensibility, flexibility and adaptability. Possible interactions in these scenarios will be presented in the next two sections.

With the scenarios defined, we developed an implementation of the framework for testing its performance in the two scenarios. As this was simply a prototype, designed for testing purposes on specific scenarios, it is important to notice that some of the framework features, which were included for further extensibility, flexibility and adaptability were neglected in this implementation for simplicity purposes. The implementation will be presented in detail in the rest of the chapter.

4.1 Scenario 1: Learning the Colors

In this scenario, a robotic agent is taught different colors by a human teacher and should, after that, be able to identify those colors when prompted. Furthermore, after learning a given color, the agent should be able to select the object (ball in our example) of the correct color when prompted for the one of the given color, amongst multiple choice possibilities.

A possible interaction for this scenario is the following:

USER: What color is this?
[*User shows a yellow paper sheet to the robot.*]

PHASTER: Sorry, I don't know. What color is it?

USER: It's yellow.

PHASTER: Ok, now I know. Thanks!

USER: What about this? What color is it?
[*User shows a blue paper sheet to the robot.*]

PHASTER: I don't know that color. What color is it?

USER: It's blue.

PHASTER: Thanks, now I know one more color.

USER: Can you guess this color now?
[*User shows an orange paper sheet to the robot.*]

PHASTER: I believe it's yellow. Am I correct?

USER: You were close, but no.

PHASTER: Oh, what color is it then?

USER: Orange.

PHASTER: Ok, I'll remember that.

USER: Now, finally, tell me what color this is.
[*User shows a blue paper sheet to the robot.*]

PHASTER: Oh, I'm sure that's blue!

USER: Great!
[*User places three balls of different colors in front of the robot.*]

USER: Find the yellow ball!
[*The robot rotates until it is facing the yellow ball.*]

PHASTER: Here it is!

USER: Nice!

4.2 Scenario 2: Acquaintances

In this scenario, a robotic agent meets multiple people and learns their names through conversation. Later on, if a known person greets the agent, it should be able to greet that person, using the correct name.

A possible interaction for this scenario is the following:

PERSON 1: Hi!

PHASTER: Hi! I'm Phaster. What's your name?

PERSON 1: Eugénio.

PHASTER: Nice to meet you, Eugénio!

PERSON 2: Hello Phaster!

PHASTER: Hi! Who are you?

PERSON 2: I'm David.

PHASTER: Hi David!

PERSON 1: Hi again!

PHASTER: Hi Eugénio!

4.3 Competences

By analyzing the two scenarios, we decided that, in order to perform as expected, the agents should have hearing, speaking, seeing and moving competences. Given this, we defined the two following predicate sets, the first defining a set of high-level actions the agents may perform and the second defining a set of high-level perceptions the agents may capture.

4.3.1 Actions

Rotate(direction, speed): Rotate the body at the given speed until it is facing the given direction.

Move(distance, direction, speed): Moves the body a given distance, in the given direction, at the given speed.

Find(object, properties): Tries to find the properties of the given object.

State(object, properties, certainty): States the properties of the given object, with a relative certainty.

Ask(object, properties): Asks for properties of a given object.

Greet([person]): Greets someone, using the person's name if it is given.

Thank([person]): Thanks someone, using the person's name if it is given.

4.3.2 Perceptions

Crash(): States that the robot has collided with something.

Color(red, green, blue): A color represented as its red, green and blue components.

Circle(x, y, radius, color): A circle represented has its center's x and y coordinates, as well as its radius and color in RGB.

Face(id): A face represented by an id calculated by the face recognizer. The reasons for this kind of representation will be explained later.

Statement(object, properties): A statement about the properties of an object.

Question(object, properties): A question about the properties of an object.

Request(object, properties): A request for an object with the given properties.

Greeting(): A greeting.

Negation(): A negation(e.g. no) negating something.

Affirmation(): An affirmation(e.g. yes) confirming something.

4.4 Brainiac

While designing our Brainiac implementation, we though about using OWL ontologies (Section 2.4.1) for knowledge representation and Pellet (Section 2.4.4) as a reasoner and, thus, we decided to develop the managers in the Java programming language for a matter of compatibility. However, later, during development, we noticed that the kind of knowledge stored and reasoned with in our scenarios was too simple for the need of such an evolved reasoner and, thus, we decided to represent and reason with it in simpler ways, which will be presented later in this section.

4.4.1 Perception Manager

For the Perception Manager, we included a sub-module, the Perception Storage, which stores the asynchronously received perceptions until a new iteration of the KNOC cycle starts. When a new iteration starts, the main module gets those perceptions and processes them, selecting the important ones using the following guidelines:

- Questions are always important.
- Requests are always important.
- Greetings are always important.
- Statements are important if they have the same object and any of the properties of any of the agent's objectives.
- Negations are important if any of the agent's objectives is waiting confirmation.
- Affirmations are important if any of the agent's objectives is waiting confirmation.
- Crashes are always important.
- Colors are important if the agent has an objective with color as a property.
- Circles are important if the agent has an objective with circle or ball as object.
- Faces are important if the agent has an objective with an object related to person and name as a property.

After this process, the important perceptions are sent to the Knowledge Manager as updates to short-term memory and notifications are sent to the Objective Manager informing it of the existence of new knowledge which may be useful.

4.4.2 Knowledge Manager

As previously stated, we thought about representing knowledge as OWL ontologies, however, as in our scenarios the knowledge needed to be stored persistently is limited to associations between color codes and respective names, and between people and their names, we opted for simpler representations.

4.4.2.1 Short-term Memory

As short-term memory stores only the most recent important perceptions, we opted for storing the perceptions themselves, which are replaced every time a new perception of the same type is received.

4.4.2.2 Colors

For the representation of associations between color codes and respective names, we defined a structure called *Color Cluster*. Each of these clusters is associated with a different color name and has a representative color code, which is calculated as the average of the all the color codes associated with the color name, and a maximum deviation calculated as the Euclidean distance between the representative of the cluster and the color code associated with the color name which is further from the representative. Every time a new color code is added to a cluster, both the representative and maximum deviation of that cluster are recalculated.

When finding possible names for a given color code, the Euclidean distance between that color code and the representative of each color cluster is calculated. If that distance is lower than the maximum deviation of the cluster plus a leveling factor associated with the existing number of clusters, then that cluster is a candidate for the given color code. The candidate clusters are ordered ascendantly in distance between the color code and the cluster's representative.

4.4.2.3 People

We wanted our agents to be able to associate a name with any person they met. However, we were not able to find a face recognition system able to perform online training and, as this was not part of our focus, we opted for using a pre-trained face database, together with the Eigenfaces (Turk and Pentland 1991) algorithm for face recognition. This approach reduced the agents' learning process to a simple association between the identifier given by the face recognizer and the name obtained through dialogue.

Given this, for the representation of the knowledge obtained through the learning process, we opted for a simple hash map, mapping face recognizer identifiers into names obtained through dialogue.

4.4.3 Objective Manager

The Objective Manager generates and updates objectives according to the existing knowledge and generates action when that knowledge is not enough. For our two scenarios we defined three kinds of objective and their respective possible states:

Answer(question): Represents the objective to answer a given question.

NEW: The objective was recently generated and is still unprocessed.

WAIT COLOR: The objective needs new color information.

WAIT STATEMENT: The objective needs new information about a given object.

WAIT CONFIRMATION: The objective is expecting confirmation.

COMPLETE: The objective was accomplished.

Find(object, properties): Represents the objective to find an object with the given properties.

NEW: The objective was recently generated and is still unprocessed.

WAIT CIRCLES: The objective needs new information about the circles present around the agent.

WAIT MOVEMENT: The objective needs new information about a given object.

COMPLETE: The objective was accomplished.

Greet(): Represents the objective to greet someone.

NEW: The objective was recently generated and is still unprocessed.

WAIT FACES: The objective needs new information about people near the agent.

WAIT STATEMENT: The objective needs new information about a given object.

COMPLETE: The objective was accomplished.

Although there are only three, limited and specific kinds of objective, they are enough to achieve the expected functionality for our scenarios. This is one example of the previously referred neglections of extensibility, flexibility and adaptability for the sake of simplicity, as this implementation is just a simple testing prototype.

These objectives are generated and updated according to the received perceptions and were defined so that, in friendly conditions, only one action at most is needed for obtaining enough knowledge to change each objective's state.

4.4.4 Execution Manager

As the transition between states of the previously defined objectives requires the maximum of one action, we decided not to store the action plans and execute the actions immediately instead. This way, the Execution Manager receives an action to be executed from the Objective Manager, verifies if there is a similar one in the action history and, in the negative case, stores it in the history and dispatches it to the competence manager as a command.

4.5 Mourinho

In contrast with the complexity of the mind, the competence manager is a simple dispatcher, able to select which commands belong to each processor and to send the perceptions they capture to the mind. Given the specificity of the scenarios, this is a simple task, with the commands being dispatched as follows:

- Ask / State / Greet / Thank → Language Processor
- Find → Vision Processor

- Rotate / Move → Mapped into actions → Body

Also, input from the body sensors is redirected to the competence processors and, on the other way around, actions sent by the processors are redirected to the body.

As Mourinho is constantly processing commands and perceptions, it was developed in C++ for the sake of performance. Furthermore, the whole process is event based, with multiple threads being created for dispatching the received messages when they arrive.

4.6 Jint

For the language processor, we decided to use English as the agent's language, thus, we had to find both an English recognizer and an English synthesizer. We decided to use CMU Sphinx[1] (Lee, Hon, and Reddy 1990) for recognition and MARY TTS[2] (Schröder and Trouvain 2003) for synthesis, as they are widely used and have acceptable performance and results. As they both are easier to use with the Java programming language, we developed the Translator module for the same language.

For the Speech Recognizer module we developed a simple grammar, as explained on the CMU Sphinx website, defining the sentences we were expecting to capture in our scenarios. With this grammar loaded, the CMU Sphinx speech recognizer receives the audio input which was directed from the body to Jint by the competence manager and is able to capture the sentences matching the grammar, generating the corresponding textual representation, that can be translated into predicates by the Translator module.

For the Speech Synthesizer, it is necessary to configure MARY TTS by selecting the language and kind of voice to be used. After that it is enough to send the textual sentences received from the Translator module to MARY TTS for it to generate the corresponding audio signal. This audio signal can then be sent to the body, through the competence manager, for being played.

For the Translator module we had to develop both an English Interpreter and an English Generator, mapping the textual sentences into predicates and vice-versa.

4.6.1 English Interpreter

Due to the simplicity of the dialogue interactions in the two scenarios and the limitations imposed by the grammar developed for the speech recognizer, we opted for a bag of words approach, able to identify multiple keywords in the same sentence, generating the corresponding predicates.

4.6.2 English Generator

For the same reasons we opted for the use of a bag of words approach for the interpreter, for the generator we defined a few simple rules for the generation of English sentences from the

[1] http://cmusphinx.sourceforge.net/
[2] http://mary.dfki.de/

predicates. Notice that the *_global* object is a special object representing the absence of a specific object.

State(object, properties, certainty) – Generate a statement:

1. Generate beginning of the sentence according to the certainty of the statement.
2. If the object is not *_global* , generate its possessive case.
3. For each property:
   (a) Write the name of the property.
   (b) If the certainty is positive, write "is" plus the value of the property.
4. End with a period.

Ask(object, properties) – Generate a question:

1. Start with "What" plus "is" or "are" according to the number of properties.
2. If the object is a personal pronoun write the corresponding possessive pronoun.
3. Otherwise, if the object is not *_global* , write it.
4. Write the properties separated by comma or "and".
5. End with a question mark.

Greet([person]) – Generate a greeting:

1. Generate a random salutation.
2. If the person is known, add the person's name.
3. Otherwise generate self presentation.

Thank([person]) – Generate a thanking message:

1. Generate random thank sentence.
2. If the person is known, add the person's name.

After being generated, these sentences can be sent to the Speech Generator module for being converted into the corresponding audio signal.

4.7 vPro

As the learning focuses of our scenarios are colors and people's names, we decided that the best way to capture both color and face information that could be associated with linguistic information was through vision, that is, by capturing visual features in the video captured by a camera. To achieve this, we added a new competence processor to our architecture for the vision competence. This vision processor, vPro, as can be seen on figure 4.1, consists on an agglomerate of feature detectors which receive the video frames and detect possible elements of the corresponding feature in those frames.

Once again from figure 4.1 we can see that vPro also has a Communicator module with the same function as in the other components. It receives commands from the competence manager, from which it extracts the video frames, which are sent to the feature detectors relevant for

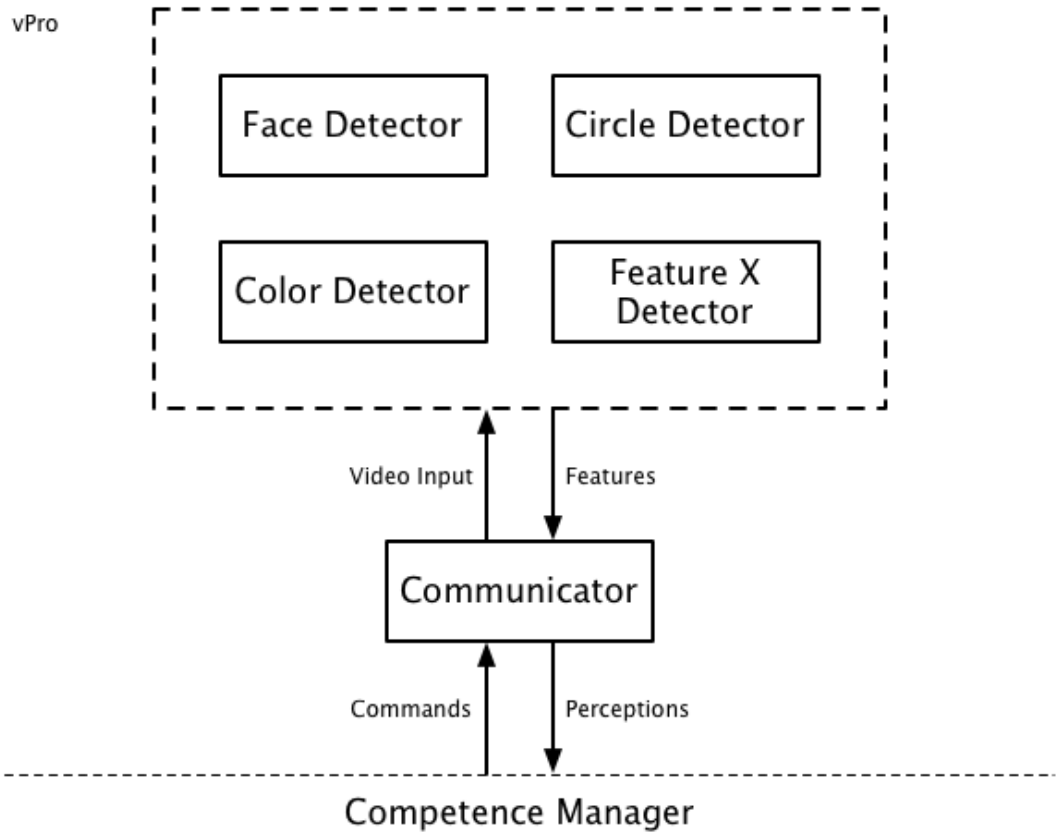

Figure 4.1: vPro: The Vision Processor

the respective commands. For our scenarios we decided to include detectors for colors, circles and faces, however, vPro is prepared for the simple addition of new ones. These detectors analyze the video frames and capture the corresponding features if present. These features are sent to the Communicator module, which sends them to the competence processor as perceptions.

Similarly to the competence manager, the vision processor also has performance issues that must be thought of, as it must quickly process video frames to find features. So, we also opted for the C++ programming language to implement it. Furthermore, the OpenCV[3] (Bradski 2000) library already provides multiple feature detectors that can be used as is, or combined to produce more evolved detectors, and has a C++ version, thus, we decided to take advantage of its capabilities while implementing vPro.

Specifically about our three detectors, using the functions of the OpenCV library we were able to capture the color of a given zone in the frame, using the RGB average of the pixels; circles using the *Hough Circle Transform*; and faces using a *Haar Cascade*. As these detectors are not part of the focus of our work, they were developed in a quick and simple way, which led to some imperfections that partially limited some instances of our scenarios.

[3] http://opencv.willowgarage.com/

4.8 Body

As a body, we decided to use Magabot[4], a platform designed to give mobility to laptops, as together with a laptop it provides us the sensors we need – camera, microphone, and crash bumpers – as well as the actuators – speakers and wheels. Furthermore, it has the capability to support other structures on top of it, so that height can be adjusted according to the scenario. The physical appearance of our agent can be seen on figure 4.2.

Figure 4.2: Phaster, our learning robot

Using this body's sensors and actuators, together with both competence processors, we were able to implement the Body Abstraction Interface with the previously presented required competences.

4.9 Communication

We did not present the Communicator module for any of the components, as the same approach used was the same for every component. Using the YARP[5] (Metta, Fitzpatrick, and Natale 2006) framework we were able to obtain a simple, asynchronous, and inter-language communication interface.

For each possible connection between the components two ports were created, one on each end of the connection, and connected to each other, creating the connection between the two components. Using these ports and the *Bottle* object, which is virtually able to transport any kind of object, anything can be sent from one component to the other. However, on the receiver side, an event is triggered every time a *Bottle* is received and, thus, its content may be processed for validity checking before being sent to the other modules.

[4] http://magabot.cc
[5] http://eris.liralab.it/yarp/

4.10 Configuration

Finally, for configuration, we decided to take advantage of the Java programming language Properties for the components written in that language, while for the components written in C++ we developed our own configuration file reader. The configurations are written in the form *object.property = value*, as can be seen on this example, showing a possible configuration for Jint:

```
# Jint Properties
# Author: Eugenio Ribeiro

# Main properties
jint.language = en
jint.synthesizer = mary
jint.recognizer = sphinx
jint.communicator = yarp

# YARP properties
yarp.portName = /Interpreter

# MARY properties
mary.server = localhost
mary.port = 59125
mary.locale = en_US
mary.voice = cmu-slt-hsmm
mary.effects = TractScaler(amount:1.25;)+
mary.audioType = WAV_FILE
mary.outputFile = output.wav

# Sphinx properties
sphinx.config = resources/sphinx.config.xml
```

4.11 Summary

This chapter presented the two relatively simple test scenarios we defined for our framework. The first one about learning the colors and the second about meeting people.

For these scenarios, it was defined that the agents should have speaking, hearing, seeing, and moving competences. Given this, a set of possible actions and perceptions was defined.

Furthermore, this chapter presents our prototype implementation of the framework for the two scenarios.

Starting from the top, on Brainiac, as the knowledge dealt with in the scenarios is too simple to require representation as an OWL ontology, we opted for simpler representations like clusters and hash maps. This also led to the definition of just three simple kinds of objectives that are, however, able to define the whole behavior of the agents in these scenarios.

As for the competence manager, it was developed as a simple dispatcher, able to select which commands belong to each processor and to send the perceptions they capture to the mind.

For the language processor, we decided to use CMU Sphinx[6] (Lee, Hon, and Reddy 1990) for speech recognition and MARY TTS[7] (Schröder and Trouvain 2003) for speech synthesis. Also, we developed our own English – Predicate / Predicate – English translator using simple rules.

We decided that the best way to capture both color and face information from the environment was through vision, so, we developed a vision processor, vPro, which is basically an agglomerate of feature detectors able to capture those visual features in video frames. For our two scenarios we developed three of these detectors, for colors, circles, and faces, by taking advantage of the OpenCV[8] (Bradski 2000) library, which provided us means for a simple image processing.

As a body, we decided to use Magabot[9], as together with a laptop it provides us the sensors and actuators we need.

Finally, we took advantage of the YARP[10] (Metta, Fitzpatrick, and Natale 2006) framework for communication and of language specific properties for configuration.

The evaluation of the two test scenarios will be presented in the next chapter.

[6] http://cmusphinx.sourceforge.net/
[7] http://mary.dfki.de/
[8] http://opencv.willowgarage.com/
[9] http://magabot.cc
[10] http://eris.liralab.it/yarp/

5 Evaluation

"You can't. He must stand trial."
– Anakin Skywalker in Star Wars Episode III: Revenge of the Sith

In order to evaluate each scenario, we performed some objective tests that evaluate the agent's performance in that scenario, as well as subjective tests where observers give their opinion about the agent's learning skills.

There were 65 observers, with ages ranging between 17 and 40 years, with most of them assuming to have a close relationship with technology.

After observing each scenario the observers answered two questions:

1. Do you think that the robot was able to learn?
2. Do you think that the dialogue was natural?

5.1 Scenario 1: Learning the Colors

For the first scenario we used the following evaluation procedure:

1. Teach five different colors to the robot using sheets of paper.
2. Count the number of turns until the agent can guess them all correctly.
3. Ask the color of five different objects with known colors.
4. Count the number of correct answers.
5. Place three balls of different colors in front of the robot.
6. Ask the agent to select the ball of a given color.
7. Verify if the agent selected the correct ball.

By repeating this procedure ten times, we obtained the results presented on figure 5.1. As we can see, only one time more than two learning turns were needed for the agent to be able to correctly guess all the colors, which we believe are good results. Furthermore, when the agents was prompted about the color of different objects, the average number of correct guesses was 4.2 out of 5, never failing more than two guesses, which we consider to be very good results. As for the results of the ball selection test, they are not presented as they were inconclusive,

	Learning Turns	Correct Guesses
Round 1	1	4
Round 2	2	5
Round 3	2	4
Round 4	2	5
Round 5	1	3
Round 6	1	4
Round 7	3	5
Round 8	2	3
Round 9	1	4
Round 10	2	5
Average	1.7	4.2
Maximum	3	5
Minimum	1	3

Figure 5.1: Scenario 1: Test results

due to problems with the circle detector usage, which led to no balls being detected on most of the tests. However, on the few relevant tests, the agent was able to select the right ball.

An early stage prototype for this scenario was publicly presented on a robotics board part of the Game On[1] exposition. Although this was a very simple prototype and it still did not include speech recognition, the public reacted very well to it, understanding what we were trying to show and proposing multiple applications outside the color domain.

Furthermore, the public stated that if robots were able to learn through dialogue with open domain, the number of common people wanting to intellectually develop their own robots would highly increase, as there would be no need for knowledge of complicated programming languages. However, it was also stated that speech recognition was a must for this kind of robot, as it is much simpler for humans to speak than to write, leading to a much more pleasant experience for the human users.

As for the observer's opinion, as can be seen on figure 5.2, 92% of the observers believe that the robot was able to learn. However, the other 8% believe the contrary, with reasons ranging from believing that the agent was just randomly guessing, to the difficulty of defining what might be considered learning.

As for the dialogue being natural, as can be seen on figure 5.3, 88% of the observers agreed, while the other 12% disagreed, stating that the vocabulary was limited and that it will always be like that, as it must be programmed.

5.2 Scenario 2: Acquaintances

For this scenario we expected to use an evaluation procedure involving at least two people, however, due to the previously referred limitations of the face recognizer and lack of availability from possible users, we had to use the following single person procedure:

1. The person greets the agent.

[1] http://gameon.gameover.sapo.pt/

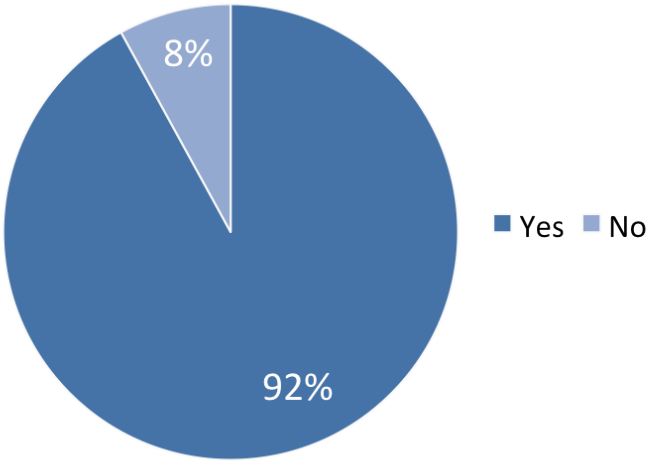

Figure 5.2: Scenario 1: ”Do you think that the robot was able to learn?”

2. When prompted, the person introduces himself to the agent.
3. The person leaves the room.
4. The person comes back and greets the agent.
5. Verify that the robot remembers the person's name.

This procedure was repeated ten times and in all of them the agent was able to notice the presence of a person and remember the correct name.

For the question about the robot's learning skills, the results were slightly lower than the ones from the previous scenario, as can be seen on figure 5.4, with only 89% believing that it was able to learn. The lower value is due to the fact that there was only one person interacting with the agent and, thus, the observers were unable to understand if it could distinguish between two different people.

However, as can be seen on figure 5.5, for the dialogue question, the results were slightly higher than on the previous scenario, with 89% of the observers stating that the dialogue was natural. As reasons for not agreeing, the only one different from the ones stated in the previous scenario was that the agent did not show emotional affection towards the person.

5.3 Summary

In this chapter we presented the evaluation of our two test scenarios, both objective and subjective.

For the first scenario, learning the colors, the results were very good with the agent only once needing more than two teaching turns to learn all the colors and with an average of 4.2 correct guesses out of 5 possible when prompted about the color of different objects. Furthermore,

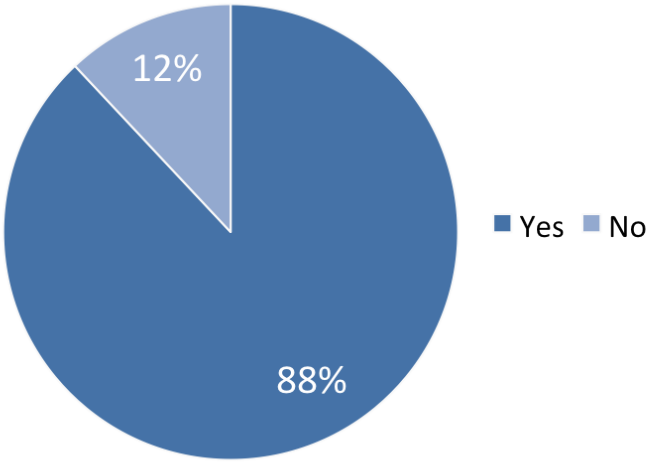

Figure 5.3: Scenario 1: ”Do you think that the dialogue was natural?”

92% of the observers stated that the agent was able to learn and 88% stated that the dialogue was natural. However, due to problems with the circle detector, the ball selection tests were inconclusive.

For the second scenario, acquaintances, the results were also good, with the agent noticing the presence of a person and remembering the correct name in every test round and 89% of the observers stating the dialogue was natural. However, as the scenario involved just one person, the observers were not as sure about the agent’s learning capabilities, with only 89% believing that it was able to learn.

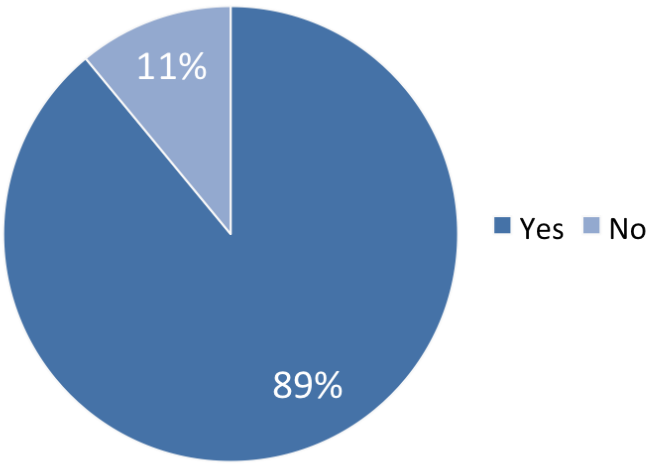

Figure 5.4: Scenario 2: ”Do you think that the robot was able to learn?”

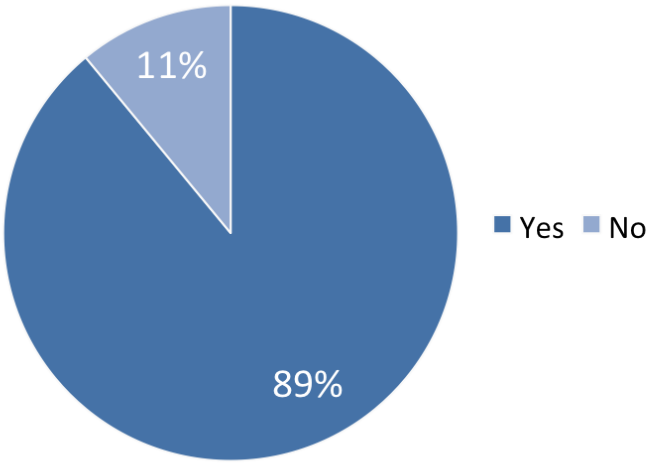

Figure 5.5: Scenario 2: ”Do you think that the dialogue was natural?”

6 Conclusions

"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning."

– Winston Churchill

6.1 Conclusions

From the previous chapters we can conclude that agents developed using our framework are able to learn through dialogue, at least in relatively simple scenarios.

As it was developed with focus on extensibility, flexibility and adaptability it is simple to adapt the framework to multiple scenarios, through the addition or extension of actions, perceptions and competence processors.

Furthermore, in our test scenarios, the performance was good, with fast action and response times. Also, the accuracy in the evaluation tests was really high.

As for the opinion of the observers, it was positive for the large majority. However, there were also some negative opinions, which can help us improving our system in the future.

Continuing on the negative side, some of the vision processor's detectors did not work as expected. Furthermore, speech interpretation and generation, as well as knowledge management, were not very developed and scenario specific, which, in some ways, limited the developed agents. However, we believe that this is normal, as for the development of such a large framework in such a short time, some parts of its deployment must be neglected.

In our opinion, we were able to fulfill the expectations, at least partially and we believe that the previously presented results show that the great majority of the observers is of the same opinion.

6.2 Future Work

In the most immediate future, we want to improve the vision processor, by correcting the circle detector and developing a face recognizer able to perform online training.

Also, in the near future we want to improve speech interpretation and generation, as well as knowledge management, in order to allow more open scenarios and dialogues.

Furthermore, we want to test the framework in more complex scenarios to be aware of its true value. One of the scenarios we are more eager to develop is one where the agent is able to learn information about itself and use that information to perform new actions.

On a different area, we would also like to add emotional components to our system, as we believe it can make the dialogue more natural and the agents more appealing to the humans interacting with them.

Bibliography

Akerkar, R. and P. Sajja (2010). *Knowledge-Based Systems* (1st ed.). Jones and Bartlett Publishers.

Akinobu Lee, T. K. and K. Shikano (2001). Julius - an open source real-time large vocabulary recognition engine. In *European Conference on Speech Communication and Technology (EUROSPEECH)*, pp. 1691–1694.

Allen, J., G. Ferguson, and A. Stent (2001). An Architecture For More Realistic Conversational Systems. In *6th international conference on Intelligent user interfaces*, pp. 1–8.

Bechhofer, S. (2003). The DIG Description Logic Interface: DIG/1.1. In *DL2003 Workshop*.

Bohus, D., A. Raux, T. K. Harris, M. Eskenazi, and A. I. Rudnicky (2007). Olympus: an open-source framework for conversational spoken language interface research. In *HLT-NAACL 2007 workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology*.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Christensen, H. I., G.-J. M. Kruijff, A. Sloman, and J. L. Wyatt (2010). *Cognitive Systems* (1st ed.). Spinger.

Dias, J., S. Mascarenhas, and A. Paiva (2011). FAtiMA Modular: Towards an Agent Architecture with a Generic Appraisal Framework.

Gruber, T. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition 5*(2), 199–220.

Kawamoto, S., H. Shimodaira, T. Nitta, T. Nishimoto, S. Nakamura, K. Itou, S. Morishima, T. Yotsukura, A. Kai, A. Lee, Y. Yamashita, T. Kobayashi, K. Tokuda, K. Hirose, N. Minematsu, A. Yamada, Y. Den, T. Utsuro, and S. Sagayama (2002). Open-source Software For Developing Anthropomorphic Spoken Dialog Agents. In *PRICAI-02, International Workshop on Lifelike Animated Agents*, pp. 64–69.

Kawamoto, S., H. Shimodaira, T. Nitta, T. Nishimoto, S. Nakamura, K. Itou, S. Morishima, T. Yotsukura, A. Kai, A. Lee, Y. Yamashita, T. Kobayashi, K. Tokuda, K. Hirose, N. Minematsu, A. Yamada, Y. Den, T. Utsuro, and S. Sagayama (2004). Galatea: Open-source Software for Developing Anthropomorphic Spoken Dialog Agents. In *Life-Like Characters: Tools, Affective Functions, and Applications*, Cognitive Technologies, pp. 187–212. Springer-Verlag.

Kruijff, G.-J. M., M. Janiček, I. Kruijff-Korbayová, P. Lison, R. Meena, and H. Zender (2009, July). Transparency in situated dialogue for interactive learning (in human-robot interaction). Technical Report DR 6.1, DFKI GmbH, Saarbrücken.

Larsson, S. and D. Traum (1999). Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkitl. *Natural Language Engineering Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, 323–340.

Lee, K.-F., H.-W. Hon, and R. Reddy (1990). An Overview of the SPHINX Speech Recognition System. *IEEE Transactions on Acoustics, Speech, and Signal Processing 38*(1), 35–45.

Martins, F., J. P. Pardal, L. Franqueira, P. Arez, and N. J. Mamede (2008). Starting to Cook a Tutoring Dialogue System. In *Spoken Language Technology Workshop*, pp. 145–148.

Martins, F. M. F. (2008, September). DIGA - Desenvolvimento de uma plataforma para criação de sistemas de diálogo. Master's thesis, Instituto Superior Técnico.

Metta, G., P. Fitzpatrick, and L. Natale (2006). YARP: Yet Another Robot Platform. *International Journal on Advanced Robotics Systems 3*(1), 43–48.

Mival, O. and D. Benyon (2007). Introducing the Companions Project: Intelligent, Persistent, Personalised Interfaces to the Internet. In *The 21st British HCI Group Annual Conference (HCI 07)*.

Ortony, A., G. L. Clore, and A. Collins (1988). *The Cognitive Structure of Emotions* (1st ed.). Prentice Hall.

Paiva, A., J. Dias, D. Sobral, R. Aylett, S. Woods, L. Hall, and C. Zoll (2005). Learning by Feeling: Evoking Empathy with Synthetic Characters. *Applied Artificial Inteligence 19*(3), 235–266.

Pelachaud, C., V. Carofiglio, B. D. Carolis, F. de Rosis, and I. Poggi (2002). Embodied Contextual Agent in Information Delivering Application. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp. 758–765.

Russell, S. and P. Norvig (2002). *Artificial Intelligence: A Modern Approach* (2nd ed.). Cambridge University Press.

Schröder, M. and J. Trouvain (2003). The German Text-to-Speech Synthesis System MARY: A Tool for Research, Development and Teaching. *International Journal of Speech Technology 6*, 365–377.

Sirin, E., B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz (2007). Pellet: A Practical OWL-DL Reasoner. *Web Semantics 5*(2), 51–53.

Skocaj, D., M. Janicek, M. Kristan, G.-J. M. Kruijff, A. Leonardis, P. Lison, A. Vrecko, and M. Zillich (2010). A basic cognitive system for interactive learning of simple visual concepts. In *ICRA 2010 Workshop ICAIR - Interactive Communication for Autonomous Intelligent Robots*, pp. 30–36.

Teng, Z., Y. Liu, and F. Ren (2009). An Integrated Natural Language Processing Approach for Conversation System. *World Academy of Science, Engineering and Technology* (51).

Tsarkov, D. and I. Horrocks (2006). FaCT++ Description Logic Reasoner: System Description. In *International Joint Conference on Automated Reasoning (IJCAR 2006)*, pp. 292–297.

Turk, M. and A. Pentland (1991). Eigenfaces for Recognition. *Journal of Cognitive Neuroscience 3*(1), 71–86.

Wooldridge, M. (2002). *An Introduction To MultiAgent Systems* (1st ed.). John Wiley and Sons.

Zovato, E. and M. Danieli (2008, April). The Companions Project: Results from the First Year. Technical Report 1, Loquendo.