A Scalable Mixed Initiative Dialogue Manager

Presenting a Novel State Representation and Dialogue Design Method for the Spoken Dialogue System of the Romeo Robot

Masters Thesis Cognitive Artificial Intelligence

Author: Roland Meertens, s3009653

Supervisors:
Dr. Louis Vuurpijl
Department of Artificial Intelligence
Radboud University, Nijmegen
Dr. Axel Buendia
CEO SpirOps, Paris

PROJET ROMEO2

SPIR.OPS artificial intelligence

March 2015
Radboud University
Nijmegen

Abstract

The French project Romeo2 develops a robot (called Romeo) that assists elderly people in their home. In the Romeo2 project, Romeo and a human talk with each other. This spoken interaction by Romeo is facilitated with a spoken dialogue system, which is yet to be developed. Using the Romeo2 documentation we created the following list with requirements for a spoken dialogue system operating in this situation (see Section 1.3):

Requirement 1: Romeo should understand natural language.

Requirement 2: Mixed initiative: both the user and Romeo can switch to a new topic.

Requirement 3: A close mapping between situations and the dialogue.

Requirement 4: Maintainability.

Requirement 5: Smalltalk.

Requirement 6: Must fit the SpirOps framework.

Existing dialogue architectures that meet some of the requirements are described (Chapter 2), and it is specified which requirements they satisfy in particular. As no existing architecture satisfies all requirements we created a new spoken dialogue system.

Part of a dialogue system is the dialogue manager, which determines what Romeo says given the situation he is in. For this thesis we developed a novel dialogue manager, its novel aspects are:

Integration in the framework used by the Romeo project partners. On this shared platform partners "publish" information of their components. Designers of the dialogue can react to events fired on this platform, and use the shared information. This closes the loop between the perception components on this platform and the interaction[37].

Representation of the dialogue state. The dialogue is divided into "topics", and the dialogue system is able to quickly switch between these topics. Each topic has an activation to indicate its relevance. Each second an activation function orders the topics to see which topic is the most relevant. The activation of each topic is determined using the speech input of the user, memories that change on the shared platform, and events on the shared platform.

How designers create dialogues. Designers create "drives" in the SpirOps editor. Each drive describes what output must be given to the actuators of the robot, and what the "motivation" and "opportunity" of this drive are. The drive with the highest combined motivation and opportunity passes its output to the actuators. Advantages of our representation are that it is easy to add drives to a dialogue, maintain the dialogue by editing separate drives, and the ability to merge the dialogue of two designers.

During an internship at the company SpirOps we implemented the dialogue system. We concluded that our dialogue architecture satisfies all requirements for the Romeo2 project. It is possible to design a dialogue that follows the scenarios in which Romeo should function, as given in the Romeo2 documentation. After a comparison to the program Choregraphe we concluded that both programs have a lot of functionality in common, but there are several differences. As dialogues created with our dialogue manager are scalable, and it is possible for the user to give topic-unrestricted input, our dialogue manager is better for the Romeo2 project than Choregraphe.

Contents

Chapter 1

Introduction

It is generally accepted that robots will be introduced in the homes of elderly people[3, 22, 48]. One project that focusses on realising this is project Romeo2 that develops a robot called Romeo. Romeo helps elderly users in their own home. In addition to performing household tasks, this robot communicates with the user. Romeo should be able to have a spoken conversation with his users, because elderly people prefer interaction through familiar means[48]. To facilitate the spoken interaction between Romeo and his user a dialogue system must be developed, a non-trivial task with many challenges which we will describe in this thesis [10, 20, 54].

In this thesis we describe the developed dialogue system of Romeo. One part of the dialogue system is the program that determines what the robot will say given the situation it observes: the dialogue manager. The three challenges in dialogue management[20] for the dialogue system we focus on are:

Integration of information All Romeo2 partners publish information on a shared platform[37]. This means a lot of information can be used in the dialogue. Selecting what information should be used, and what information is not important for the dialogue, is still a challenge.

Initiative Deciding the topic of the conversation is a still a challenge[20]. When modeling a conversation in a traditional dialogue system the designers indicate when topic-switches can occur (where the initiative to switch to another topic is either taken by the system or by the user). Designing such a conversation is a time-consuming task, especially for dialogues in large domains[20, 54]. As Romeo operates in a large domain, this means a solution to this problem must be found.

Maintainability and scalability The dialogue of the Romeo robot will be large, as it will cover a lot of topics. While Romeo is deployed he will encounter situations which were not anticipated, for which a response has to be designed. With current dialogue managers maintaining the dialogue is a time-consuming process[20, 35], and it is hard to scale a dialogue to larger domains[48].

In this thesis we present a novel dialogue manager with a novel dialogue state representation that is scalable and can process all information from its perception components.

Figure 1.1: Romeo interacting with its user. In the left image it recommends a television program to his user. In the right image Romeo brings breakfast to its user[19].

1.1 Project Romeo2

The goal of project Romeo2 is a social service robot for elderly people. Romeo assists the user with simple tasks to ensure that the user can live autonomously for a prolonged period. In Figure 1.1 several examples can be seen of what Romeo should be able to do. An important part of the project is the dialogue system of the robot, the development of this system is the topic of this thesis. More information about project Romeo2 can be found at `http://projetromeo.com/`.

The Romeo robot is developed by the company Aldebaran. It is a humanoid robot with a height of 1.40 metres. Functionality of the robot includes walking, supporting an elderly user while walking, and picking up and carrying small objects. The Romeo robot uses the same software as the other humanoid robot of Aldebaran: the Nao robot. During the development of the dialogue system the Romeo robot was not yet available, instead we used the Nao robot. As no Romeo-specific functionality was needed for the project this did not raise any issues.

Project Romeo2 is a collaboration between 16 companies and universities in France. Partners in the Romeo project are Aldebaran, SpirOps, INRIA, ALL4TEC, CNRS-LAAS, VOXLER, CRNRS-LIMSI, CNRS-LIRMM, CEALIST, Collège de France, Armines-ENSTA, ISIR, Telecom ParisTech, Université de Versailles, Strate and Approche. Project Romeo is a project financed by Île de France.

All partners in project Romeo2 collaborate on the shared platform called NAOqi. With this unified framework all perception software components are brought together[37]. Each component developed by a partner publishes data on this platform, and all components on this framework can use the data of other components. As described by Pandey et al. components use the information of other components to create more meaningful data[37]. By providing the other components with information about the dialogue the sensing-interaction loop can be closed[37].

An overview of 40 perception components that are being developed in the Romeo2 project is given in Figure 1.2. Two examples of how information of other components can be integrated in the dialogue of the Romeo robot are:

- If the People Presence component detects that somebody enters the room Romeo can say "hello" to this person.

- If the Object Tracker recognises an object Romeo can ask a question about this object.

Each of the components on the shared platform can be used in the dialogue. Choosing what information to use is one of the challenges in the Romeo2 project. Section 3.2.1 describes our approach to this challenge.

(I) Perception of Human	
(i) People Presence	(ix) Perspective Taking
(ii) Face Detection	(x) Emotion Recognition
(iii) Face Characteristics	(xi) Speaker Localization
(iv) Gaze Analysis	(xii) Speech Recognition
(v) Face Recognition	(xiii) Speech Rhythm Analysis
(vi) Face and Person Tracking	(xiv) User Attention Detection
(vii) Posture Characterization	(xv) User Profile Analysis
(viii) Waving Detection	(xvi) Intention Analysis

(II) Perception of Robot Itself
(i) Battery Status
(ii) Body Temperature
(iii) Foot Status
(iv) Robot Posture
(v) Fall Detection
(vi) Self Collision Detection

(III) Perception of Object	(IV) Perception of Environment	(V) Perception of Stimuli
(i) 3D Segmentation	(i) Landmark Detection	(i) Sound Detection
(ii) Barcode Reader	(ii) Darkness Detection	(ii) Chest Button Interpretation
(iii) Close Object Detection	(iii) Place Recognition	(iii) Movement Detection
(iv) Object Recognition	(iv) Location Tracker	(iv) Sound Localization
(v) Object Tracker	(v) Sound Tracker	(v) External Collision Detection
(vi) Semantic perception	(vi) Semantic Perception (place)	(vi) Contact Observer

Figure 1.2: An overview of 40 perception components that are created for the Romeo2 project[19]

1.2 SpirOps

The practical work of this thesis was performed during an internship at the company SpirOps. This company is a private scientific research lab focused on Artificial Intelligence [2]. One of their products is a design paradigm called "Drive Oriented". With this paradigm behaviours can be created by incrementally adding small pieces of code using a graphical editor. Each behaviour indicates how relevant it is at that moment, and the most relevant behaviours are executed. Designer creates a "SpirOps brain", the collection of all behaviours. The SpirOps brain executes the decision process of a program. We used the "Drive Oriented" paradigm in the creation of the dialogue, this is further explained in Section 3.5.

The tasks SpirOps has in the Romeo2 project are targeted at developing a representation of the world around Romeo (called the world model), the dialogue system, and the decision making module. During my internship the work on the world model and decision making module had not yet started. This meant that it was not yet possible to incorporate the world model and decision making capabilities of Romeo in the dialogue system. In Section 4.2 we describe how this affects the dialogue possibilities. This section also describes how we partially solved this problem by creating a relevant world model ourselves, and how the SpirOps world model will be used in the future.

1.3 Problem setting: scenarios and theory

In this section we discuss the requirements of the dialogue system for project Romeo2. At the start of the internship we set the requirements by looking at the following sources:

- The scenarios Romeo will encounter. The dialogue system should minimally be able to work in these scenarios to be useful for project Romeo2. What scenarios Romeo encounters is described in Subsection 1.3.1.
- The theories about dialogue management when interacting with elderly people. Interacting with elderly people is different from interacting with younger people[48]. What the implications are for the requirements of the dialogue system can be found in Subsection 1.3.2.

1.3.1 Scenarios

There are five scenarios in the requirements document of the Romeo2 project in which Romeo must be able to function[19]. As a courtesy to the reader we have translated three scenarios, they can be found in Appendix A. The two untranslated scenarios describe:

- How a paraplegic man uses Romeo by commanding him to pick up books and bringing them to him. It must be possible to ask Romeo to pick up things, or hold things, during a conversation. One example of a phrase the user utters is: "pick up that book". To make this scenario possible it is important that Romeo can both talk and control its arms at the same time, which is multi-modal interaction. At the end of this section we explain how we deal with multi-modal input and output.
- That Romeo is asking the user questions about the user. In this scenario Romeo invites users to engage in more social interaction. Smalltalk is an important aspect in the Romeo2 project, and a non-trivial task[28]. How we implemented smalltalk is found in Section 4.2.

As can be read in these scenarios Romeo has to perform a lot of tasks. In the translated scenarios Romeo sometimes interrupts tasks, to continue this task at a later moment. One example where this happens is scenario 1-scene 2 in Appendix A. While Romeo is helping the user getting ready for the day, the caregiver arrives. Romeo greets the caregiver, makes breakfast, plays music, and then continues with helping the user get ready for the day. Romeo also interrupts the user, to indicate that he should take his medication. The task of the robot can change very quickly, and both the user and the robot sometimes change the topic of a conversation. The user as well as the robot should be able to take the initiative in the conversation; the dialogue system should be mixed-initiative.

To show that our developed dialogue system can indeed be used in these situations we will describe what we implemented in this thesis. Once Romeo is deployed it will encounter more scenarios in which it must be able to function. This makes it important that the dialogue is maintainable and scalable. In Section 5.2 the implementation of the scenarios is described.

Dialogue systems are evolving towards multi-modal interaction[9, 39]. The requirements of the Romeo2 project state that in this project the focus is spoken interaction. We will discuss in what way our system can be used for multi-modal interaction in Section 3.5.

1.3.2 Dialogue management

There are already multiple approaches to dialogue management[9, 24, 25, 34]. One important challenge in dialogue management is that designing the dialogues is a time-consuming process[20]. Although research attempts to solve this problem using machine learning techniques[61], these approaches can not scale yet to the range of possible dialogues Romeo should have[48]. A thorough explanation on why using machine learning techniques is not possible in project Romeo is given in Section 2.1.1.

In general dialogue systems for elderly people differ from dialogue systems for younger people[22, 38, 42, 48]. It is harder for elderly people to adapt to a new dialogue interaction method than for younger people. Because of this the interaction modality in the Romeo2 project is speech. It has been shown that older users are less likely to speak to a system in a way that is easy for the system to understand[22]. Romeo should thus understand natural language.

As described in Section 1.3.1 the dialogue will be extended while Romeo is deployed. Romeo must have a response for as many situations as possible. If there are problems with a certain situation it is important designers can solve these problems quickly. This makes it important that:

There is a close mapping between situations and the dialogue A close mapping between the problem world (the situations Romeo encounters) and the design world (what is designed to solve these problems) makes it easier for developers to solve problems with the dialogue[23]. This requires a good dialogue representation, which we describe in Section 3.2.

The dialogue is easy to maintain It should be easy to add parts of a dialogue to make the dialogue scalable. There will be multiple designers working on the dialogue. All designers will work on a specific dialogue part, while we want the resulting dialogue to cover a large amount of situations. If two designers create a different dialogue these two should be easy to combine in a new dialogue covering more topics.

As described in Section 1.2 the internship was conducted at the company SpirOps. This company has several years of experience building systems that use artificial intelligence. In previous projects they already built dialogue managers that are able to take multiple sources into account[11]. To continue on this work we set as a requirement that our dialogue system must fit in the existing SpirOps framework.

1.4 Requirements

The previous section mentions what the requirements of the dialogue system should be, given the scenarios of the Romeo2 project and dialogue theory. The requirements are:

Requirement 1: Romeo should understand natural language As it is often difficult for elderly people to use a new interaction style, spoken interaction in natural language is essential [48].

Requirement 2: Mixed Initiative Mixed initiative: both the user and Romeo can switch to a new topic.

Requirement 3: A close mapping between situations and the dialogue The dialogue must be so robust that it is possible to use the system directly on real users. To achieve this

there must be a close mapping between situations and the dialogue[48, 61].

Requirement 4: Maintainability New dialogues should be easy to program and easy to maintain. This includes the possibility to merge dialogues easily.

Requirement 5: Smalltalk It must be possible for designers to create a smalltalk dialogue.

Requirement 6: Must fit the SpirOps framework It should be possible to integrate the dialogue system in the existing SpirOps software.

1.5 Research questions

In Section 1.3 and 1.4 we described the problem setting and introduced requirements for the dialogue system. This leads us to the first research question:

1. How to design a dialogue system that meets all six requirements described in Section 1.4?

As we will show in Chapter 2 there is no existing dialogue system that meets all requirements. We approach this research question by starting to build a dialogue system that integrates with the existing SpirOps components. When the dialogue system is created it is important to evaluate the performance of our dialogue system. This leads to the second research question:

2. How does one evaluate a spoken dialogue system?

We approach this question by looking at existing dialogue systems and how they were evaluated. In Section 5.1 the most relevant evaluation methods are listed, including what the disadvantages of each evaluation method are. Based on these options we will select the evaluations that are the most relevant for our system and the Romeo2 project. Once the second question is answered we start the evaluation of our developed dialogue system. In this evaluation we focus on how designers create dialogues with our system, and how our dialogue system will be used in the future of the Romeo2 project. This leads to the final research question:

3. Is our developed dialogue system usable to design the dialogue of the Romeo2 project?

1.6 Outline

This thesis is organised as follows. A background on dialogue systems, and the most relevant existing dialogue systems are presented in Chapter 2. For each system we list which requirements in Section 1.4 are met, and which requirements are not met. Our dialogue system is described in Chapter 3. Each of the components in the system is described, and the novel approach to dialogue management is clarified. Chapter 5 describes what evaluation methods we considered, and how we performed the most relevant methods. Chapter 6 describes the results of this evaluation. In Chapter 7 we conclude that our dialogue manager is suitable for the Romeo2 project. In the same chapter we make recommendations for future dialogue managers.

Chapter 2

Existing dialogue systems

In the introduction we discussed the requirements of the Romeo2 dialogue system. This chapter discusses the tasks of such a system, how dialogue is managed, and what dialogue strategies are important. In this chapter we discuss several existing applications and existing research as well as examine in what way they satisfy the requirements set in Section 1.4.

2.1 Theory spoken dialogue systems

Spoken dialogue systems can be divided into three main categories [30]:

Graph based Dialogue systems guide the user through a dialogue that consists of a finite set of predetermined states. Generally the system holds the initiative, the computer asks questions and based on the answer of the user a new state is chosen. In most graph based systems the user is given a choice of options that are relevant for the selection of the next state. To optimise the state selection process these systems usually constrain the user in what he can say[30].

Frame based Dialogue systems try to fill slots in a template. These dialogue systems try to execute tasks (for example: query a database) for which they need information from the user. By asking the user for specific information these slots can be filled, and the task that needs these slots can be executed. These systems are able to extract information from the input sentence of the user to fill the form. An advantage of frame based systems over graph based systems is that users can supply various bits of information simultaneously through speech. An example is: "What time does the bus between Nijmegen and Amsterdam leave?". In this sentence the city where the user wants to depart, the city where the user wants to arrive, and the method of travel specified. This is usually not possible in graph based systems.

Agent based Dialogue systems are designed to create more complex communication between the system and the user. The system and user are both seen as agents who are capable of reasoning about their knowledge and their actions. Systems in this category tend facilitate mixed initiative, and the processing of speech input is not constrained by the previous output of the system.

Agent based dialogue systems have the properties we want our dialogue system to have, as described in Section 1.3.2. In the introduction of Chapter 3 we elaborate on our choice of an agent based system for the dialogue system of project Romeo2.

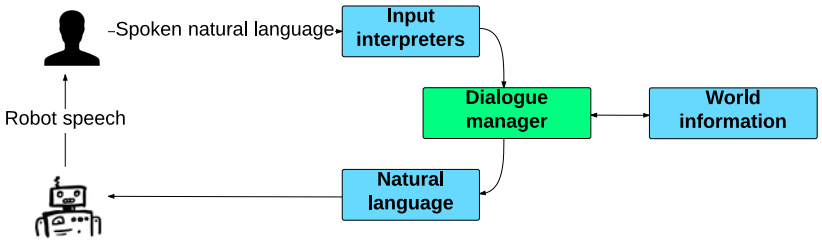


Figure 2.1: A simplified overview of a dialogue system. Spoken input of the user is interpreted and sent to the dialogue manager. The dialogue manager uses the information of the input interpreters and the world information to decide what to say. This decision is sent to the natural language generators, which determines the choice of wording by the robot.

Part of a spoken dialogue system is the dialogue manager (see Figure 2.1). The functions of the dialogue manager are[56]:

- Updating the dialogue state. The dialogue state contains the information used that is necessary to create relevant output. Examples of what can be included are: the topic of the conversation, the last utterance of the user, and the last speaker[24]. Based on what happens around the robot, the dialogue state should be updated.
- Providing expectations about what the user might say.
- Interfacing with the domain. For the Romeo2 project this means using the NAOqi platform described in Section 1.1.
- Deciding what to say, and when to say it.

Of course the dialogue system for Romeo2 needs a dialogue manager that performs all of the above functions. There are several approaches to dialogue management, we will discuss them in the next subsection. In Section 3.2 the novel dialogue state of the dialogue system is described, as well as how it is updated. In Section 3.5 the last function of the dialogue manager is described: the output decision.

2.1.1 Dialogue management

How to design what the dialogue manager has to do in each situation is still a challenge for spoken dialogue systems[20]. In Section 1.3.2 we shortly discussed the requirements of the dialogue manager. There are two directions in dialogue generation: hand-crafted by human designers, or automatically generated using machine learning algorithms[61]. The advantage of using a machine learning approach is that designers do not have to spend time designing dialogues. Dialogue management approaches that use machine learning are:

- Reasoning via types of plans[16, 25, 45]. The system creates a plan with what states the user should be able to reach, and what states should be followed to get there. Each time the user gives input the system determines in what way this input contributes to each of the possible goals, and what transition to a new state should be made to reach this goal.
- Logical inference[30, 51], which can be used to determine what information is or is not available to the user. When the system knows what information the user wants to know it can query its database and give a response to the user.
- Statistics[45], such as POMDPs[61] that model the "state-transition probabilities". Given the likelihood that the system is in dialogue state A, they try to determine the probability that the user wants to go to state B.
- Reinforcement learning[27, 53], which can be used to determine what the dialogue policies are for an information system.
- Neural networks [31, 32, 59, 60], which are used to predict what task-specific dialogue acts are currently used by the actors of a dialogue. This can be used to design a response for each dialogue act the user uses, for each situation.

Unfortunately there are severe downsides to each of the machine learning approaches[48, 49, 61]:

- They can not deal with larger and more complex domains[48, 61]. Current approaches are tested on small domains with simple functions, such as planning systems or information retrieval systems. When there are more possible functions in more complex domains these approaches are impractical[61]. To support this we quote Young et al. who recently wrote: "Standard POMDP methods do not scale to the complexity needed to represent a real-world dialogue system"[61].
- The sheer volume of data required, where data sparsity is a huge problem[49]. To support this we quote Young et al. who recently wrote: "Bayesian network approaches have the advantage that they can model a rich set of conditional dependencies and can be trained on data, although once again data sparsity is a major problem"[61]. Although user tests have been conducted in the Romeo2 project this data was not yet available during my internship. This makes using current state of the art machine learning techniques not an option yet for the Romeo2 project.

Although machine learning techniques have improved considerably in the last years, we chose to let designers create the dialogues. This offers the robustness and performance in large domains that machine learning techniques currently lack. In Section 3.2 we describe our novel approach to a dialogue state representation, and in Section 3.5 we describe our novel approach to dialogue management. In these sections we will argue that our dialogue manager with hand-crafted dialogue can deal with large complex domains, and is scalable (contrary to current state of the art machine learning approaches to dialogue management).

2.1.2 Conversational strategies

An important part of a dialogue system is its conversational strategy[29]. The conversational strategy is the combination of strategies the system uses in its conversation with the user. Examples of strategies are: "when does the system speak?" and "how can the user take control of the dialogue?". No requirements for the conversational strategy were specified in the Romeo2

projects, and no requirements have been set regarding the conversational strategy in Section 1.4. However, it is important to discuss our conversational strategy as it affects how the user communicates with the robot. The strategies we consider in this thesis are the same strategies the developers of Ravenclaw considered[9]:

Grounding During conversations, the user and the dialogue system need to have the same mutual knowledge, beliefs and assumptions[12]. This is called the "common ground" which both partners use for a more efficient dialogue. The process of reaching the common ground is called "grounding". An example of this process is that conversational partners tell their conversational partner what knowledge they have. Both participants in a conversation keep track of the knowledge of their partner. Grounding is still a challenge for conversational systems, with no standard solution[27, 33]. The challenge of grounding is not giving too much information (as the conversation will get boring) and not omitting vital details (as the user and system will fails to reach the common ground).

Turn-taking During the conversation the system and user speak alternatively. The initiative to start a conversation by talking can be with the robot, with the user, or mixed. As with grounding there is no standard solution to turn-taking[4]. One part of a turn-taking mechanism is barging in, this makes it possible for the user to interrupt the dialogue system while it is speaking. This is an optional mechanism that some dialogue systems support[10], and others do not[30, 34].

Universal dialogue mechanisms Some dialogue systems offer mechanisms for the user to let him control the dialogue system. Examples can be uttering "help", "repeat", "suspend"[9], "what can I say" and "start over"[7]. These systems help the user reach his goals in an efficient way. Some systems include communications about the dialogue itself to the dialogue mechanisms, such as "thanking", or "acknowledging"[24]. Implementing these mechanisms universally for all dialogues eases the creation of dialogues for the dialogue designers.

In Section 3.5.2 we discuss what conversational strategies are implemented in our dialogue system.

2.2 Dialogue design methods

In this section we describe five prominent dialogue systems. These systems all satisfy part of the requirements given in Section 1.4, but unfortunately none of them satisfies all requirements.

2.2.1 SpeechBuilder

SpeechBuilder facilitates the creation of mixed initiative spoken dialogue systems[21]. Designers define what actions and concepts the system can understand. When the user says something the speech recogniser will try to extract an action and the concepts in the sentence. Examples of input SpeechBuilder understands can be found in Figure 2.2.

The dialogue is written down in a markup language (XML) document. Designers define what the system will answer for each action the user might ask the system to execute.

turn on the lights in the kitchen **action=set&frame=(object=lights,room=kitchen,value=on)**
will it be raining in Boston on Friday **action=verify&frame=(city=Boston,day=Friday,property=rain)**

Figure 2.2: Example input speechbuilder understands. In each sentence the action is extracted that the user wants the system to perform. The concepts in the sentence are also extracted, so the system is able to execute a query in their database.

Upsides	Downsides
• The system understands natural language. • The dialogues are mixed initiative. • There is a close mapping between situations and the dialogue. • It is possible to program a smalltalk component. • It is possible to fit SpeechBuilder in the SpirOps framework. This is due to the modular approach of SpeechBuilder[21].	• The system is hard to maintain: the dialogue module does not have the possibility to merge multiple dialogues. Each new possible input should be added to the XML files.

Table 2.1: Up- and downsides of SpeechBuilder

2.2.2 VoiceXML

VoiceXML is used to develop a spoken dialogue system over the telephone. By using the markup language XML the designer is able to create a program that can ask the user questions. For each answer of the user, the designer will have to specify what the system does. A dialogue in which the system says "hello world" is displayed below. An example of a dialogue with input from the user would take at least 144 lines. The length of even a single dialogue indicates that the maintainability of a VoiceXML dialogue is low. However, the maintainability can be improved by using a visual development tool.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version = "2.1" >
<form>
<block>
<prompt>
Hello World. This is my first telephone application.
</prompt>
</block>
</form>
</vxml>
```

Upsides	Downsides
• The system understands natural language. • There is a close mapping between situations and the dialogue. • It is possible to create a smalltalk component.	• It is hard to create and maintain programs due to the representation of the dialogue. This problem can be solved by using a visual XML editor. • It is hard to create a mixed initiative dialogue. • It is not possible to fit VoiceXML in the SpirOps framework.

Table 2.2: Up- and downsides of VoiceXML

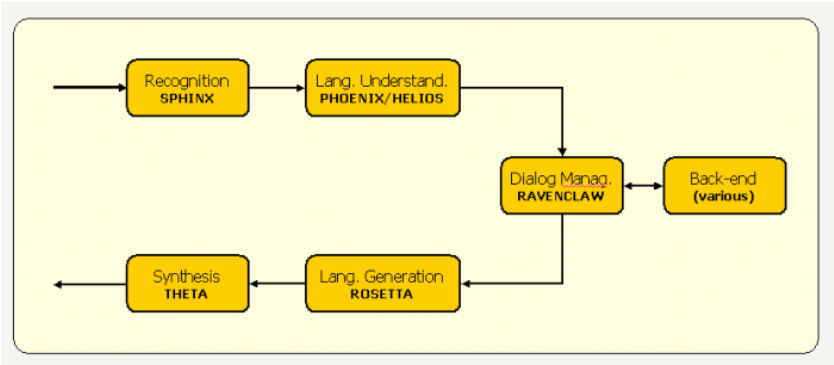


Figure 2.3: The pipeline of Olympus. This image is taken from the site `https://www.cs.cmu.edu/~dbohus/ravenclaw-olympus/what_is_olympus.html`

2.2.3 Olympus

Olympus is a spoken dialogue architecture developed at the Carnegie Mellon University. It can be deployed over a phone line. Olympus has proven itself to be useful, as several operative dialogue systems were designed using Olympus[8]. It is still possible to talk with several of the implementations by calling their numbers with a phone.

Examples of projects that have been implemented with Olympus are:

- Roomline: A telephone based system allowing users to reserve conference rooms within one of the faculties at the Carnegie Mellon University `http://www.cs.cmu.edu/~dbohus/ravenclaw-olympus/roomline.html`
- Let's Go! Public Bus Information System [43]: A telephone based system built in 2005 providing callers access to bus routes and scheduling information in the greater Pittsburgh area `http://www.cs.cmu.edu/~dbohus/ravenclaw-olympus/letsgopublic.html`
- LARRI: A multi modal system providing aircraft personnel assistance during maintenance tasks `http://www.cs.cmu.edu/~dbohus/ravenclaw-olympus/larri.html`
- Madeleine: a text-based dialogue system for medical diagnosis designed for the MITRE dialogue workshop challenge `http://www.cs.cmu.edu/~dbohus/ravenclaw-olympus/madeleine.html`

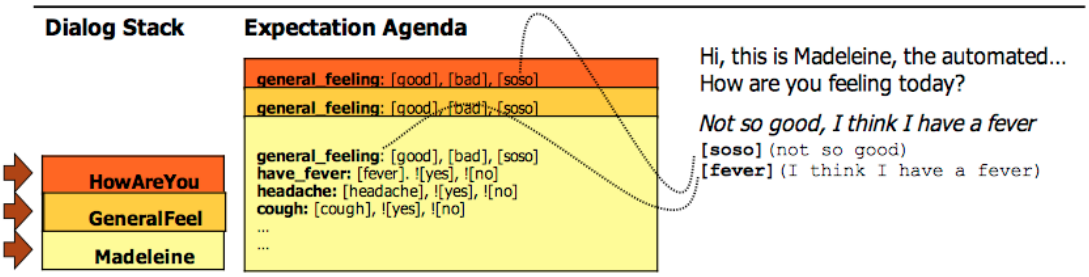


Figure 2.4: An example of a dialogue stack in the Ravenclaw architecture. This image can be found in the slides about Madeleine, a system using Ravenclaw, at `http://www.cs.cmu.edu/~dbohus/docs/madeleine.ppt`

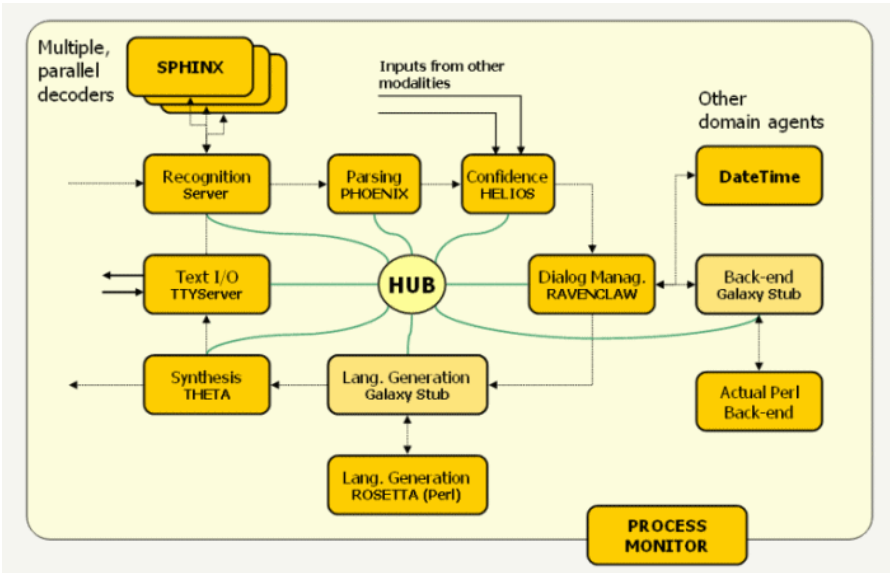


Figure 2.5: The components of the Olympus architecture. This image is taken from the site `https://www.cs.cmu.edu/~dbohus/ravenclaw-olympus/what_is_olympus.html`

The dialogue manager of Olympus is Ravenclaw: this dialogue manager is a successor to the Agenda architecture which was developed in 1999 as a telephone based dialogue manager[50]. The dialogue is represented by a tree of dialogue agents, where each agent handles a part of the dialogue. The dialogue is generated by traversing through the tree from left-to-right depth-first. Turn-taking is used to create execution phases and input phases, the dialogue stack is consulted to see in what phase the system should be. Sub-dialogues can be handled by the system, as an agent is able to add multiple dialogue-agents to the stack. The stack with the current chosen dialogue agents is used to construct the input expectations of the system. A visualisation of this stack is displayed in Figure 2.4.

The language processing component selects information from the users' speech input it anticipated. What the system expects to hear from the user is called the "expectation agenda" in

Ravenclaw. The input from the user is bound to concepts by doing a top-down traversal of this agenda. Each node in the expectation agenda is a "handler" [50], each handler specifies a form of receptors corresponding to input nets. An input net tries to find all words belonging to a certain category. For example: the "date" handler tries to match input of the user to a specific date, while the "username" handler tries to match input of the user to known usernames. When the user gives input the expectation agenda is traversed. As soon as part of a sentence can be matched to a handler, this part of the sentence is marked as "consumed" and will not be matched to other handlers.

Upsides	Downsides
• The system understands natural language. • There is a close mapping between situations and the dialogue. • It is possible to design a smalltalk component.	• Dialogues are hard to design and hard to maintain. Experienced designers are able to setup a system within one month, and can fine-tune it in another month (see `www.cs.cmu.edu/~dbohus/docs/build_w_ravenclaw.ppt`). It is impossible to easily merge several dialogues. • Mixed initiative: It is possible for the user to ask some questions to the system. Unfortunately it is impossible to ask questions to the system that are not related to the current dialogue state. • It is not possible to fit Ravenclaw in the SpirOps framework.

Table 2.3: Up- and downsides of Olympus

2.2.4 Choregraphe

Choregraphe is a program developed by Aldebaran to design programs for their robots[1, 41]. Examples of programs that can be created with Choregraphe are:

- A motion the robot performs
- A simple program (for example: picking up a glass)
- A dialogue with the user

Programs are represented as flowcharts and programmed using a graphical user interface (see Figure 2.6).

The designer of a Choregraphe program places boxes with actions, operators, or values. These boxes are connected by events, and can give integers, strings or other data-objects as arguments to each other. It is also possible for programs to react on NAOqi events.

A linear program, where the user has few options and the robot takes the initiative, is easy to design with Choregraph. This is because the linear notation of Choregraphe is close to the problem setting[23]. An example of a simple part of a dialogue is shown in Figure 2.6. As soon

as programs become bigger it is harder to maintain them. Choregraphe is still being actively developed, but with the current version (version number 1.14.1) designing and maintaining long dialogues takes a long time.

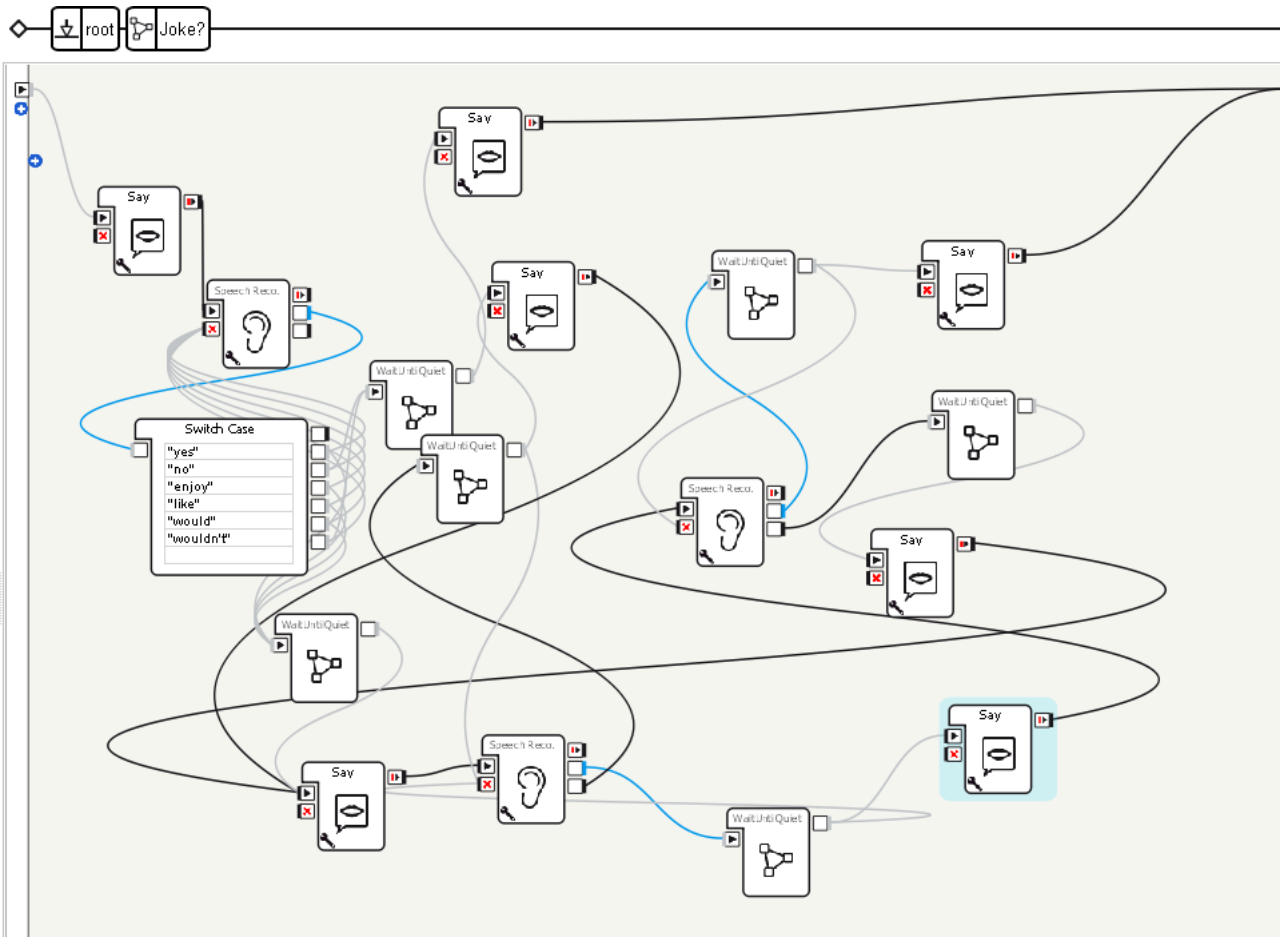


Figure 2.6: An example of a small Choregraphe script. The robot asks the human whether he wants to hear a joke. Based on the answer of the human the robot either tells a knock-knock joke, or does not tell a joke

Upsides	Downsides
• There is a close mapping between the situations and the dialogue[6]. • It is easy to program a smalltalk component.	• The system does not understand natural language. Although the robot is able to spot words in a sentence the user has to know what words he can use. • Maintainability: although small dialogues are easy to program maintaining and creating bigger programs is very difficult. It is impossible to merge two different programs. • Mixed initiative: although it is possible to add options for the user to get the initiative these programs are hard to create and maintain. • It is impossible to fit Choregraphe in the SpirOps framework.

Table 2.4: Up- and downsides of Choregraph

2.2.5 Re-phrase

Re-phrase is an online modular editor created by the company MindAffect. Designers use this program to create dialogues for the program cChat(`http://www.mindaffect.nl/?page_id=85`). The program cChat facilitates the interaction between two humans (of which one has problems with talking). Currently it is not yet possible to design the interaction between a human agent and a dialogue system. An example of a use case is the interaction with an ALS patient. These patients often have trouble talking, but with cChat they can interact with visitors. Each dialogue turn the human agent can select one of the four sentences that are applicable in that situation. The answer of the other user can be answered, but it is also possible to pose a question to the other person, or propose to start a new activity.

Re-phrase offers a drag and drop interface, this should enable everybody to create dialogues. Dialogues are modular, from each part of a dialogue a link can be made to another dialogue. The online editor helps in distributing the design of dialogues among multiple designers. This makes the scalability of dialogues created with Re-phrase a particular strength of Re-phrase.

Upsides	Downsides
• There is a close mapping between the situations and the dialogue. • Dialogues are easy to maintain due to the online collaboration approach (although it is impossible to merge dialogues easily). • It is possible to design smalltalk.	• There is no mixed initiative: each dialogue turn it is only possible to select one of the few available options. • The system understands natural language, only one of the few available options can be selected. • It is impossible to fit Re-phrase in the SpirOps framework.

Table 2.5: Up- and downsides of Re-phrase

2.3 Summary

In this chapter we discussed the essentials of spoken dialogue systems, and looked at several existing dialogue systems. Unfortunately none of the discussed existing dialogue systems is suitable for our domain, as none of them satisfies all requirements described in Section 1.4. To our best knowledge there is no existing established technology that satisfies all requirements. In the remaining part of this thesis we discuss the development of a new dialogue system. As we will discuss in Section 6.1 this newly developed dialogue system does have all the requirements described in Section 1.4.

This new dialogue system uses some of the technologies discussed in this chapter, and we will refer back to the relevant section when called for. Three examples of properties of existing systems we use in our dialogue system are:

- Both Choregraphe and Re-phrase have a close mapping between the situations and the dialogue due to their visual representation. We also use a visual editor, the SpirOps editor, to create dialogues (see Section 3.5).
- SpeechBuilder matches input on actions the system can perform, and extracts the concepts in the sentence (see Section 2.2.1). This allows a designer to create mixed initiative dialogues, and makes it possible to understand natural language. Our dialogue system uses a speech recognition technique that gives output similar to the output of SpeechBuilder (see Section 3.3).
- The Ravenclaw dialogue manager understands natural language by using the Agenda Architecture. The natural language processing component in our dialogue system is based on the Agenda Architecture. Dialogues created with the Ravenclaw dialogue manager are not mixed initiative, as it is not possible to ask questions to the system that are not related to the current topic. By adjusting the Agenda Architecture it is possible to ask questions about every topic Romeo knows about. This can be found in Section 3.4.

Chapter 3

The dialogue system

In Section 1.4 six requirements were listed which our dialogue system should meet. In Chapter 2 several existing dialogue systems were discussed, and we concluded none of them satisfy the requirements for the Romeo2 project. In this chapter the blueprint of our dialogue system is described.

Current literature shows that a modular approach should be taken for dialogue systems[8, 21]. This enables us to create components that use existing software. Instead of replacing the complete system it can be improved by replacing components. In Section 2.1 the three categories of dialogue systems were described. In that section it was stated that an agent based system is normally used for more complex communication. Other desirable properties of agent based systems are that they allow mixed initiative interaction, and that speech input is not constrained by the previous output of the system. Given these properties it was decided to create an agent based system. Every module in our system is an agent which receives input from another agent (or from the user) and provides output to another agent (or the user). Each agent determines what to do with the input it receives, and what to give as output[34].

3.1 Dialogue architecture

Figure 3.1 shows the flow-diagram of our dialogue system. Modules in our system communicate using the JavaScript Object Notation (JSON). It is possible to substitute components (for example: substitute the speech recognition system with a better speech recognition system), as long as they use the same JSON format. In this section we will mention each of the components, and further explain their details of each in the subsequent sections of this chapter.

- Speech recognition: this component determines what intention the user has when speaking to the robot. The designer can choose between two possible speech recognition components:
  - Wit: A speech recognition engine that processes the audio on a server(Section 3.3.1).
  - ALSpeechRecognition: the speech recognition component already installed on Romeo; it processes the audio without an external server (Section 3.3.2).
- Natural language processing: this component updates the dialogue state using the speech input (Section 3.4).

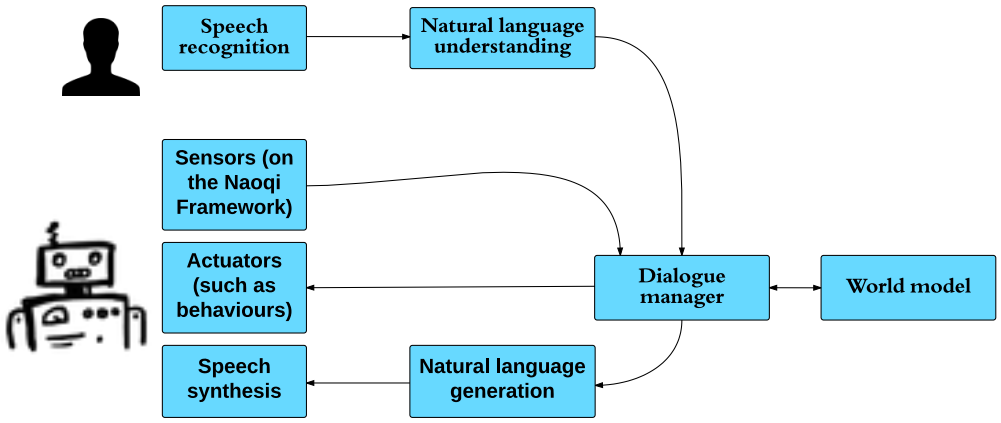


Figure 3.1: The flow diagram of our dialogue architecture

- Dialogue management: this component updates the dialogue state using data on the NAOqi platform(described in Section 3.2), and generates output for the robot (that is passed to the natural language generation module). The designer specifies for each scenario in which the robot should give output which actuators the robot has to control, and what these actuators have to do. In Section 3.5 it is described how the designer designs a part of the dialogue.
- Natural language generation: this component takes the output of the dialogue manager, and creates natural language that will be given to the speech synthesis component (Section 3.6).
- Speech synthesis: this component takes the natural language generated by the natural language generator, and transforms it into soundwaves(Section 3.7).

3.2 Novel dialogue state representation

The dialogue representation creates a distinction between the possible situations Romeo can encounter. The existing dialogue state representations we studied approached the dialogue state in two different ways[25, 56](see Section 2.1.1):

- What conversational information is known to the system. This includes who the participants of the dialogue are, the common ground (see Section 2.1.2), and a model with information about the user[25]. This representation is useful in frame-based dialogue systems (see Section 2.1).
- What states the user can reach. Especially in graph-based and frame-based systems the transitions between nodes in the graph, and transitions to new frames, are stored in the dialogue state (see Section 2.1). Based on the input of the user a transition is chosen and the dialogue state is updated.

Literature shows that there are multiple problems with these traditional dialogue representations. In this thesis we will consider a subset of these problems, the two problems Larsson et al. and

Nguyen et al. mention in their work[25, 34]. These problems are:

- Maintaining a topic-hierarchy, i.e. the rule-sets that determine what transitions to new states can be made, is difficult. This poses problems in larger domains[20], such as the domain in which Romeo should function. Designers have to think about all possible moves that can be taken at each possible state, which is a difficult and time-consuming task.
- The initiative in the conversation(described in Section 1.3.1). When modeling a conversation in a traditional dialogue system the designers indicate when topic-switches can take place (either initiated by the system or by the user). This is a time-consuming task, especially for dialogues in large domains[54]. As Romeo operates in a large domain, and one of our requirements is that the dialogue is mixed-initiative, this is a difficult problem.

To elaborate on the problems this causes we present a practical example. If Romeo is talking about the user to entertain him, the user should be able to interrupt Romeo to talk about something more important (for example: what to get for dinner). However, when it is important that the user takes his medication immediately and Romeo is instructing him to do so, the user should not be able to interrupt Romeo. In traditional dialogue systems it is impossible for the user to ask the dialogue system system a question that is not relevant to the current dialogue. This is not a problem for the small domains in which dialogue systems normally operate. However, as Romeo operates in a large domain, using the same approach to dialogue management as Ravenclaw is not possible for us.

We approach these two problems with a novel dialogue representation. This representation that uses the different topics Romeo can talk about to distinguish the situations Romeo can encounter. The dialogue state is represented by:

A list with topics This is a finite list of topics the system can talk about, specified by the designer.

The sensor values of the robot This includes button presses, distance measured with the sonars, and whether the robot detects a face.

As we will explain in Section 3.5 this representation makes it easy for designers to design the output of the robot.

Each topic is represented by the following variables. In Subsection 3.2.1 it is explained how these variables are used to update the state representation:

- The name of the topic.
- An activation.
- A minimum activation it should keep.
- A list with handlers.
- A list with memories in NAOqi the topic is subscribed to.
- A list with events in NAOqi the topic is subscribed to.

Each topic has a list with handlers (colored blue in Figure 3.2), that create the capability of understanding what the user said (see Section 3.4). Handlers in our system are an adaptation of the handlers used in the Agenda Architecture[50] and in Ravenclaw[9] (see Section 2.2.3). A handler belongs to a topic, and tries to match the spoken input of the user to the input this topic is expecting. A figure showing how a handler relates to a topic is shown in Figure 3.2. Each

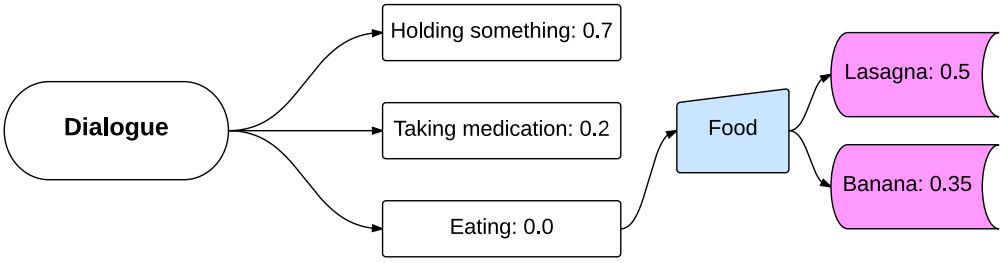


Figure 3.2: The state representation with a handler

piece of information in the spoken input that can be matched to a handler will be added to that handler. An example is the topic eating; this topic expects that the user asks for something to eat. The handler on this topic is the "food" handler, who remembers every object that is edible. In the Agenda Architecture words are "consumed" when a match is found. However, this would create problems when it is unclear about what topic Romeo is talking. We therefore decided not to consume a word when a match is found, but to add it to every relevant topic. Each handler keeps a list with matching words, with an activation for every word. How the activation is updated and how words are matched can be found in Section 3.4.

3.2.1 Activation function topics

As described in Section 1.4 one of the requirements is that the dialogue system is mixed initiative. Sheridan describes in his paper "Eight ultimate challenges of human-robot interaction " that this is still a difficult challenge[54]. In current dialogue systems the design of a dialogue is typically hand-crafted, which is a time-consuming process for mixed-initiative dialogues. The result is that traditional dialogue systems do not generalize to larger domains[20].

Our state representation is divided into several topics: these can be ordered based on their relevance at that moment. If another topic becomes more relevant the dialogue system should switch to this topic. With this representation both the user and Romeo should be able to take initiative and start talking about another topic. This introduces the following problem: how do we determine the relevance of a topic.

In Section 2.1 we described that a dialogue manager needs to interface with the domain, in the Romeo2 project this means all components of the Romeo2 partners. As described in Section 1.1 all Romeo2 partners publish information of their components on the NAOqi platform. Components in NAOqi can:

Fire an event An event indicates that something happens. Each event has a name, and components in NAOqi can subscribe to these events. For instance, there is the module "Waving Detection", with the "onWaveDetected" event. If a component subscribes to this event, whenever the robot detects that someone is waving at it this component will get a signal.

Create a memory Memories are key-value pairs stored on the robot. Components can write a value in the memory, for example: each time somebody enters the room the value stored

in the "PeoplePresent" key is changed by the People Presence component. If a component subscribes to a key value, this component will be notified when the value of that key changes.

In Figure 1.2 40 perception components that are being developed by Romeo2 partners can be seen. In Section 1.1 two examples have been given of how these components can be used in the dialogue. Providing the interface between these components and the dialogue is a challenge, we approach this challenge using the dialogue state representation.

In our dialogue manager the activation is updated using the "leaky bucket" metaphor[55]. When something relevant to a topic occurs a lot of activation will be "added" to a topic. This activation will "flow away" when the topic is not relevant for a longer time.

- By slowly deactivating each topic (called decay), the system "forgets" topics that have not been relevant for a long time.
- By re-activating topics when something relevant to these topics happens on the NAOqi platform, Romeo can take the initiative.
- By re-activating topics when the user says something relevant to these topics, the user can take the initiative.

The implication of these activation rules can be explained by the example shown in Figure 3.3. In this example, when the user wants to indicate he wants to eat a certain apple he might say "that apple". As an apple is something you can eat and hold, the topic "eating" and "holding something" will become more activated (indicated in green). The topic "holding something" was more active, so Romeo will start talking about this topic. When the user notices that Romeo is talking about holding something the user can clarify himself and say he meant eating the apple. Now the topic "eating" will become more activated, while the other topics become less activated (indicated in red). Romeo will now switch to the topic "eating", while remembering that "apple" the user said several seconds ago referred to "eating". An alternative conversation occurs when during the time the user says "that apple" the NAOqi event "takeMedication" fires. Although the topics "holding something" and "eating" will still be activated the topic "taking medication" will get the highest activation. Romeo now takes the initiative and tells the user to take his medication (or simply hands the user his medication). After the user takes his medication the topic "taking medication" becomes less relevant, and Romeo will continue the conversation the user was trying to start before.

When Romeo does not talk about a topic this topic becomes less relevant to the conversation. This means Romeo should "forget" that it was talking about this topic. Each time update t (1 per second) the system updates the activation A of each topic with an activation function. We created the following function:

$$A_t = A_{t-1} - \frac{1}{d} * A_{t-1}$$

Where the initial activation of each topic is set to

$$A_{t=0} = 0.1$$

If A_t is smaller than the minimum activation for that topic A_t is set to the minimum activation. The variable d is the decay factor, it indicates how quickly a topic is "forgotten". In our implementation d is set to 40 (at the end of this section it is explained why this value was chosen). With this value a topic whose activation is 1, will have an activation of about one third after ten seconds. The decay of an activated topic with d set to 40 can be seen in Figure 3.4.

The activation function function is non-linear, a highly activated topic will "leak" more activation than a topic with a low activation. This is a strong and novel aspect of our dialogue manager. It allows a more relevant topic to quickly take over in a conversation, and by letting it take a long time before a topic is totally forgotten a "long term memory" is represented. The system can go back to a topic it talked about before when something relevant to this topic occurs. By adjusting the value of d topics can be either forgotten faster, or be remembered for a longer time. This allows the dialogue designer to choose a value for d depending on with who Romeo will be talking. Designers can change the value of d when they believe Romeo switches to new topics too quickly, or when they believe Romeo does switch fast enough.

If something happens that makes a topic more relevant, the activation of this topic should increase. As described above the activation of a topic is influences by components on the NAOqi platform. The activation of a topic increases when a relevant NAOqi memory changes, or a relevant NAOqi event fires. When something happens that is relevant for a topic we update the activation of this topic with this formula:

$$A_t = (1 - d)A_{t-1} + d$$

In our implementation we set d to 0.5 for a memory and set d to 0.75 for an event. The difference in resulting activations can be seen in Figure 3.5. The values for the memory and event were chosen during the implementation of the dialogue walkthrough described in Section 5.2. In Section 7.3 several suggestions about changing these values to improve the dialogue system are given.

In our implementation the value of d is different for a memory and an event. Changing memories in the NAOqi framework do not always indicate a very important event. Examples are "CloseObjectDetection/ObjectInfo", that has the information about the latest detected close object. Reacting on this memory is something that could improve the conversation, but it is not necessary to react on every object the robot sees. Events indicate something important, and thus make topics that are affected by an event more relevant. An example of an event is the "PeoplePerception/JustArrived" or "PeoplePerception/JustLeft" event. Reacting on these events could improve the conversation, and we believe that reacting on this event improves the conversation more than reacting on the value change of the "ObjectInfo" memory.

The values we used for d in this chapter were chosen during the design of the scenarios (see Section 5.2). They were found by trying out several values, looking at the resulting graph of the activations to see if the right topic was selected (see Figure 6.1), and looking at the output of the robot. In Figure 6.2 an event occurs that indicates the user has to take his medication. When a higher value for d is chosen for events the topic will be switched as soon as the event occurs the first time. With a lower value of d more events are needed before Romeo takes the initiative to talk about the medication topic. As can be read in Section 6.2 no errors occurred during our evaluation that are attributed to the activation function. The activation function thus helps us achieve our goal of having a mixed-initiative dialogue, as both the user and Romeo can talk about a new topic anytime.

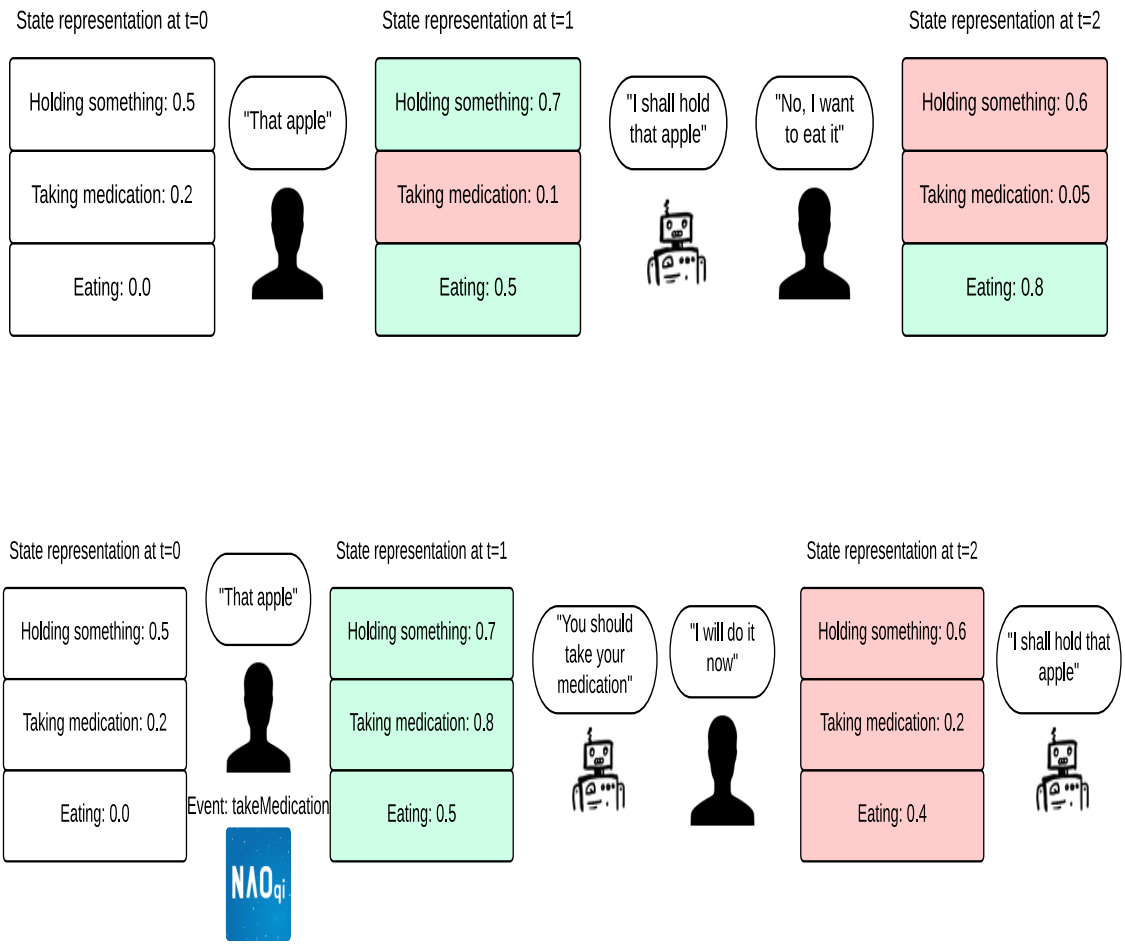


Figure 3.3: Two examples of how the dialogue system can be updated in different scenarios.

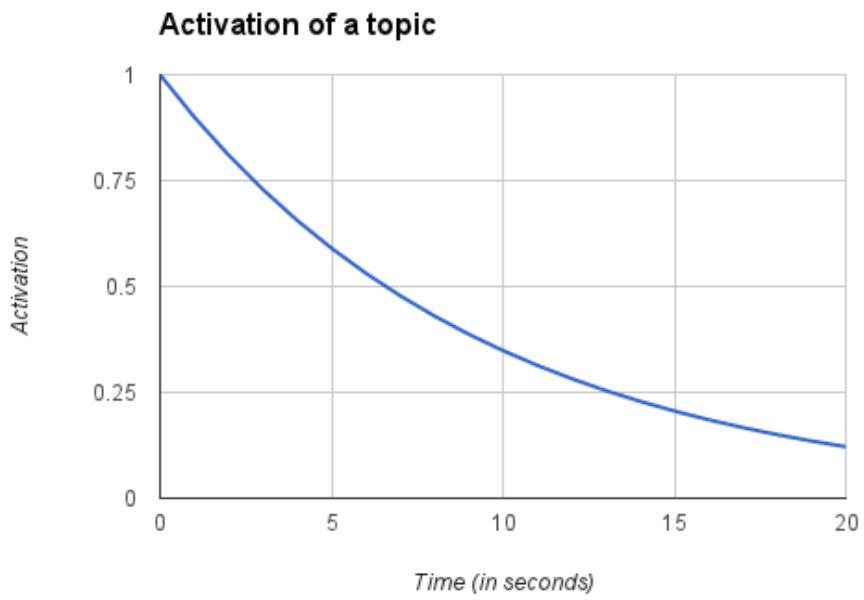


Figure 3.4: The activation of a fully activated topic over 20 seconds. The value of d is set to 40 for this example. As can be seen the function is non-linear, the topic loses its activation quickly in the first seconds, and more slowly in the last seconds.

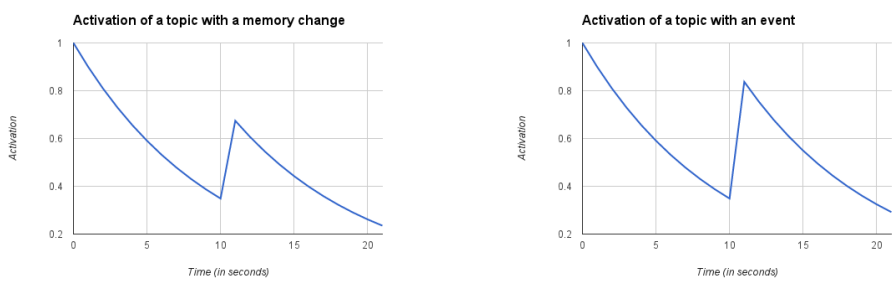


Figure 3.5: The activation of a fully activated topic with a memory change or event at 10 seconds. In this figure it can be seen that a memory update leads to a lower increase of activation than an event.

3.3 Speech recognition

The speech recognition component in our architecture uses the spoken utterances of the user to determine what the user wants to say. The output of the speech recognition component represents what the user just said, being the intent of the sentence and the entities in this sentence.

Intent An intent is what the user wants to achieve with his uttered sentence (for example: ask about the weather, set an alarm, say hello). There are multiple ways a user can express an intent (for example: "wake me up at 7am", "set alarm tomorrow at seven", "alarm at seven tomorrow morning"). The speech recognition tries to determine what was most likely the intent of the user, out of the intents supplied by the designer of the dialogue.

Entity Everything in a sentence that is important for the representation of the dialogue is called an entity. Examples are: the date and time, a specific room, the name of a person. The speech recognition tries to determine what is meant with specific words, and returns this representation and what the user uttered. A specific example found on the Wit site is that the user utters "Wake me up tomorrow at 5am", and the speech recognition finds the entity datetime (which is "2013-07-04T050000Z").

We considered and implemented two possible components handling the speech recognition: Wit and ALSpeechRecognition. One key difference is that Wit works "online" and ALSpeechRecognition works "offline". Processing natural language can require a lot of RAM memory and access to language models that require a lot of disk space. As smartphones and robots have limited memory and disk space resources a dedicated server can be used. Recorded sound is sent to the server and processed there, and the server returns a transcription of what the user uttered. A downside of online speech recognition is that it takes time to send voice recordings to the server. Using an online or offline speech recogniser is a trade-off between quality and speed. An article written by an expert on speech recognition that describes the state of the art in online and offline speech recognition can be found at `http://cmusphinx.sourceforge.net/2015/02/current-state-of-offline-speech-recognition-and-smarttv`.

3.3.1 Wit

Wit is an online speech recognition software API which processes natural language. The sounds that are recorded by the robot are processed on the server of Wit to obtain a transcription of what the user uttered. Given the raw speech or text input, Wit determines the users intent and what entities there are in the input. If the user says "Set the alarm at 7am", Wit returns that the intent is "set_alarm". It also returns the exact date and time at which the alarm should be set. With a smart concept spotting algorithm not only words can be spotted, but also concepts such as a date by saying "next friday". By adding training sentences and annotating them the Wit classifier is trained on a specific corpus. An example of how spoken input is processed can be found in image 3.6.

Wit has several features that make it more attractive than alternative speech recognizers:

- Inbox: when Wit is not sure about the resulting intent and entities it adds the input to the inbox. Here the designer annotates the string and adds it to the corpus. This feature is unique for Wit.
- Confidence: each processed input gets a confidence score that indicates how confident Wit is that it processed this input correctly.

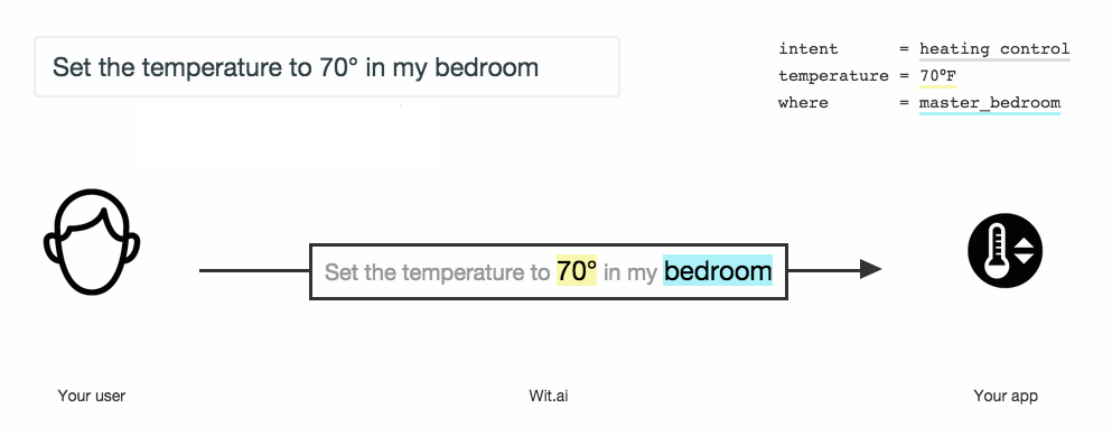


Figure 3.6: A diagram showing how Wit works. This example and more examples can be found on `https://wit.ai/`

- Multiple interpretations: Wit returns several interpretations per input and includes a certainty per interpretation.
- Written and spoken input: before determining the intent and entities spoken input is transcribed to written text. Both voice and text can be used as input modalities. While implementing the scenarios and demonstrating Wit to partners in the Romeo2 project we spoke 177 times to the robot. Although sometimes the transcription of the spoken input contained errors, Wit found the expected intent and entities 95% of the time.
- Many to one mapping: the words "yes" and "sure" both return as "affirmative" while "no" and "nope" both return as "negative". This mapping can be made for each word, and makes it easier for the designer of a dialogue to give an answer based on input of the user.
- Free of charge. Wit does not cost anything for the user and does not cost anything for the developers.

The unique features of Wit are the inbox and the fact that it gives multiple interpretations per input. Unfortunately, we also found several downsides of Wit that make this speech recognition component less useful for the Romeo2 project:

- An internet connection is required to send the speech files to their server. This might cause problems, as this means Romeo will not work when the connection is lost. During a meeting with the Wit developers they assured us there will be an offline version of Wit in the future, which solves this problem.
- There is no hands free voice activation function: the user has to press a button to speak to the robot. The Wit developers are working on this, and a solution to this problem will be released soon.
- There is a delay of about two seconds in the Wit server to process the data. When the connection between the dialogue system and the Wit server is bad the delay between speaking and recognition what the user said will be longer. A delay in the speech recognition means there will be a delay between talking to the robot and him answering, which will be annoying for the user[8]. This problem will be solved when the offline version of Wit is

ready.

- Wit has no information about the world around Romeo, as it is not integrated in the NAOqi platform. One consequence is that the user is unable to use relative clauses in his input. It is impossible for Wit to deal with pronominal references such as 'it' to indicate objects in the room. The world model could be used to find what object 'it' might indicate. As described in Section 1.2 there is no world model in the Romeo project yet. Once this model is ready it could be used to improve the speech recognition. Wit recently made their parser open-source (see `https://wit.ai/blog/2014/10/01/open-source-parser-duckling`). This makes it possible to create a new sentence parser for Wit. Note that only the parser is open source, the other Wit software is still closed source. With an adjusted parser it is also possible to take expectations into account, for example: if the user eats an apple every day it is likely that it will ask for an apple to eat. By creating a parser that takes objects around Romeo into account, users will be able to use relative clauses.

No comparative studies were performed with Wit and ALSpeechRecognition (described in the next subsection). From our personal experience using both programs extensively we have seen that Wit is a more reliable speech recognizer. We chose to optimise our dialogue system for usage with Wit because of the many attractive features. Although our language processing component (described in Section 3.4) is optimised for Wit, it can also be used with ALSpeechRecognition.

3.3.2 ALSpeechRecognition

ALSpeechRecognition is the speech recognition component that is installed by default on Romeo. The recognition is performed on the robot itself, and processing takes less than one second. ALSpeechRecognition searches for words in a corpus specified at the start of the system. The corpus has all intents and entities that were trained on the Wit website. This allows the developer to switch between ALSpeechRecognition and Wit, supporting the modular approach of our dialogue system.

An important advantage is that ALSpeechRecognition is already installed on the Romeo robot. However, ALSpeechRecognition is not a state of the art component. One downside is that the syntactic variance the user can use is low (the best recognition is achieved when uttering single words). Another downside is that in our personal experience with this component the recognition rate was low, it often chose the wrong intent when speaking to the robot (more than 50% of the time).

3.4 Natural language processing

The natural language processing component updates the dialogue state (described in Section 3.2), based on the input of the user. Input for this component is the output of the speech recognition: the intent and the set of entities of the sentence the user uttered.

Each entity consists of the word the speech recognition component recognised, and the category this word belongs to. For each entity in the input of this component we add the word to each handler that is searching for words with the same category of the category of the entity. As described in Section 3.2 each handler has a list of handled words, and an activation per word. The word the user means should have the highest activation. Romeo "learns" and "forgets" the

words the user uttered using an update function. If this handler has not captured this word before, the activation A_t of this word is set to 0.5. Every time the handler captures a new word this word must be learned, and other words must be forgotten. The activation of this word is updated using the following formula, where l is the learning rate:

$$A_t = A_{t-1} + \frac{1 - A_{t-1}}{l} * A_{t-1}$$

All other words must be forgotten slowly (deactivated gradually), the activation of the other words is updated using this formula:

$$A_t = A_{t-1} - \frac{1 - A_{t-1}}{l} * A_{t-1}$$

The value of l is set to 2 for the word that has to be learned, and to 4 for the words that must be forgotten. These values were chosen during development as this proved to work well even when the user changed his mind and wanted the handler to choose a different word. By choosing a higher value for l Romeo will learn and forget words faster. A higher value of l for learning words means that users can correct themselves at all times by saying the word they want Romeo to use one time. A lower value for learning words means that users have to repeat words if they want Romeo to understand them, but the dialogue system will be able to deal with accidental errors in the speech input. With the current activation function repeated words are getting a higher activation, while the activation of non-repeated words gets lower.

3.5 Output decision

After each update t (occurring every second) the dialogue system has to decide what output should be given. The robot has several actuators, for example: speech output, its arms, and the LED lights in his eye. The challenge of the dialogue system is selecting relevant output for each actuator in the current situation. In most current dialogue systems the design of the dialogue is hand-crafted by designers. The problem is that this can be a time-consuming process. As described in Section 1.4 it is important that new dialogues should be easy to program and easy to maintain. In that section we also say that it should be possible to merge two separately created dialogues to make one dialogue with more topics. In Section 1.3.2 we already wrote that this is a challenge, and Glass described in the paper "Challenges for spoken dialogue systems" that designing mixed-initiative dialogue systems for large domains is one of the six most important challenges of dialogue design[20]. In Section 1.4 we also described that we aim to create a close mapping between situations and the dialogue, as this way it is possible to use the system directly on real users. This section describes how we approached this challenge using the dialogue state representation described in Section 3.2.

There are existing architectures that attempt to solve decision problems for large domains by dividing them in smaller problems. One example of such a method is the subsumption architecture of Brooks et al. [47], that creates intelligent control mechanisms by layering simple behaviours. The total control is hereby fragmented into smaller units, each unit transforming input into an output. An advantage of the subsumption architecture is that it is maintainable and scalable, new behaviours can be added without affection the other behaviours. Unfortunately there are also downsides to the subsumption architecture of Brooks et al., for example: inadequate command fusion, and the inaccessibility of internal state[47]. There are already extensions to this subsumption architecture, one example is the architecture for robot control by Rosenblatt and

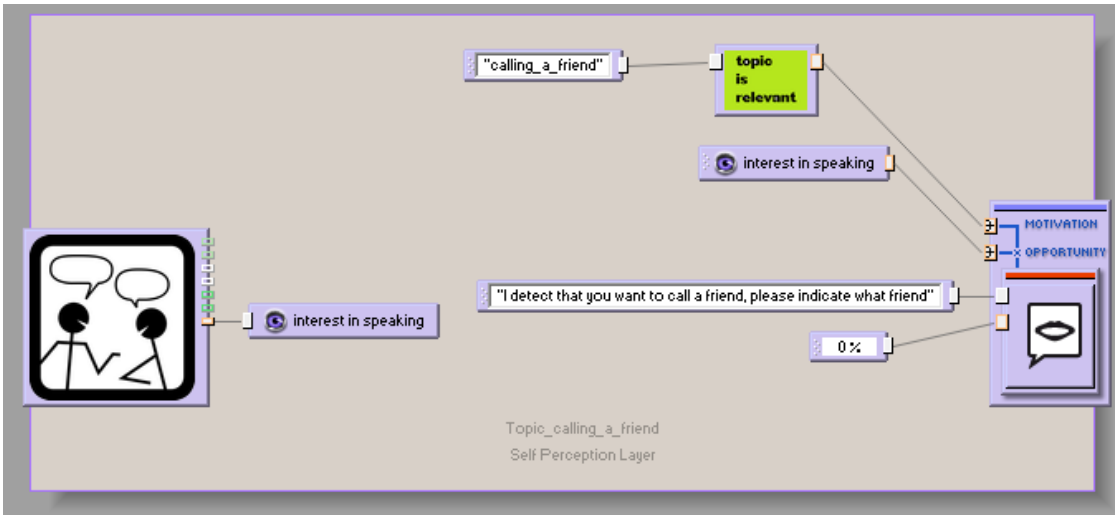


Figure 3.7: An example of a drive where the user wants to call a friend. The block on the right is where the values for the motivation, opportunity and actuators are entered. The block on the left of the image contains information about the dialogue. This includes the interest in speaking of the robot, the last sentence of the user, and the last sentence of the robot. The motivation to execute this drive is equal to the relevancy of the topic "Calling a friend". The opportunity is determined by the turn-taking component (see Section 3.5.2), being the variable "interest in speaking". The only actuator used is the speech output, and if this drive is selected the robot will utter "I detect that you want....".

Payton[47]. In their architecture each of the units indicates an activation, and the most relevant behaviours are selected. To arrive at a decision in this architecture all units indicate how relevant they are by means of an activation, and by weighing these activations the units that control the actuators are selected. One requirement set in Section 1.4 is that the decision module must fit in the SpirOps framework. Unfortunately the subsumption architecture does not fit in the SpirOps framework, which makes it not suitable for our problem domain.

Another existing solution to define output using perception components is the cognitive architecture ACT-R[5]. ACT-R is a modular component-based architecture, with modules working in parallel to create decisions. The output of ACT-R is a set of short commands, which can be used for the actuators of the Romeo robot. Unfortunately there are problems with this cognitive architecture, such as scalability[46]. As described in Section 1.4 this makes using this cognitive architecture not suitable for our problem domain.

In current research there are two approaches to modeling languages: declarative versus imperative[15]. Designing the output decision should use one of these two methods. Imperative processes represent a program as a flowchart. An example of such a program is Choregraphe, an example of a dialogue created with Choregraphe can be seen in Figure 2.6. In the program displayed here the robot first asks a question, then starts listening, and depending on the answer tells a joke or not. Declarative processes describe the logic of the program, but do not describe the control flow. An example is the "Target drives means(TDM) architecture" by Berenz and Suzuki[6]. Designers using TDM design the behaviours of the robot, and indicate what the conditions are of executing these behaviours. When executing this program the behaviours are executed as

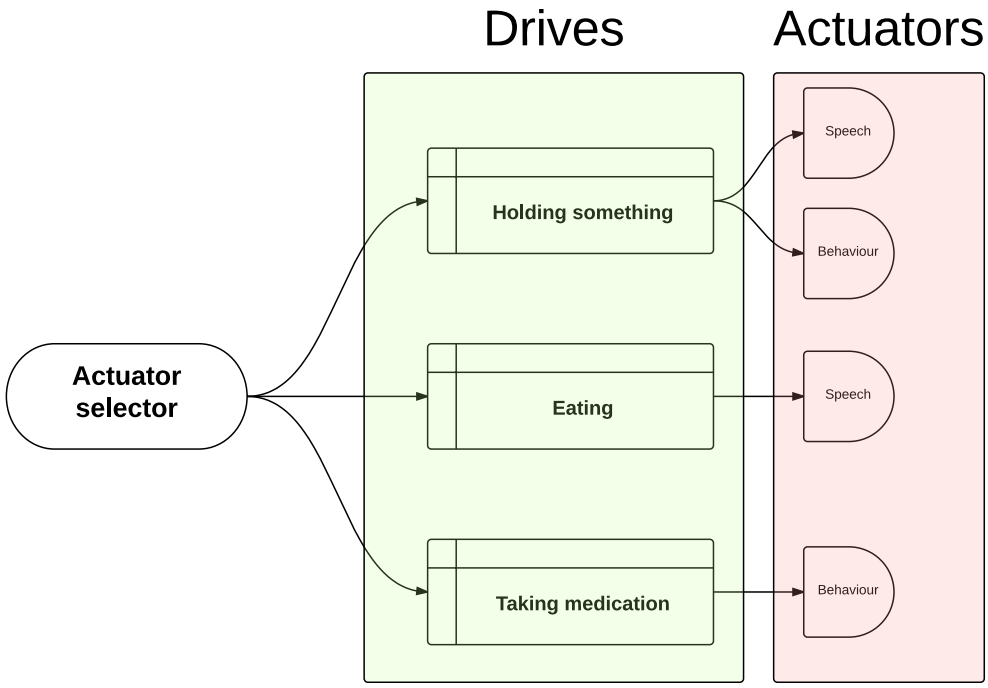


Figure 3.8: The actuator selection mechanism. Each one of the drives indicates its motivation and opportunity. For each of the possible actuators the mechanism selects the drive with the highest values. This example is explained in Section 3.5.

soon as the prespecified conditions are met. Existing literature is still divided whether declarative or imperative processes offer the best maintainability[15]. Berenz and Suzuki showed that for developers it is initially easier to learn how to design an imperative processes. However, manipulating a declarative program is less difficult at later stages[6]. As one of our goals is to create a program that is maintainable we decided a declarative design environment should be used to underpin the output decision.

For the decision module we use the SpirOps decision architecture. In Section 1.2 we explained that in this architecture a "SpirOps brain" makes decisions by choosing the relevant implemented behaviours. All behaviours in the brain are executed at the same time, but only the most relevant ones can use the actuators. Later in this section we will discuss the properties the SpirOps brain has that are beneficial for the goals of the Romeo2 project. A SpirOps brain consists of a set of "drives", each drive indicates:

Motivation How high is the motivation to give the output of this drive? If the topic that this drive belongs to is relevant the motivation is high. If the topic that this drive belongs to is less relevant the motivation is low.

Opportunity Is it possible to give output right now? If the user is speaking you do not want to interrupt him, so then the opportunity to speak has to be set to zero.

Output to actuators What actuators does this drive use? And what does the drive give as output to these actuators?

An actuator selection mechanism selects the drives that give output to the actuators. This mechanism takes the following steps:

1. Calculate the opportunity and motivation for each drive.
2. Assign the smallest value of the opportunity and motivation to each drive.
3. Select the drive with the highest value.
4. Execute this drive and assign the output of this drive to the actuators.
5. Subsequently; if there are other drives whose value is positive, and these drives are controlling actuators that have not received an output: execute one of these drives.

We use the dialogue state representation described in Section 3.2 to determine what action the robot should perform. In the dialogue representation is divided into topics, and for each topic a drive in the SpirOps architecture can be designed. Each topic needs control of a set of actuators, and these should be added to the drive of that topic. For example: the topic "eating" only needs the speech actuator to talk about what the user wants to eat that day. The topic "taking medication" only needs to execute a behaviour, and wants to do this when the user wants to get up. There are also topics with multiple actuators, such as the topic "holding something". When this topic is relevant Romeo speaks to confirm what object it needs to pick up, and executes a behaviour to actually pick up the object.

How drives define output for an actuator can be seen in Figure 3.8. In the example in Figure 3.8 the drive "Holding something" might be selected. It is also possible that both the drives "Eating" and "Taking medication" are selected, as they use different actuators. The concept behind this configuration that allows both drives to be selected is that this method ensures that the robot can talk about eating while helping the user to get up. Note that "Holding something" and "eating" can not be selected at the same time, as this would mean that both drives would control the speech actuator at the same time. If no drives have motivation or opportunity no drives are selected. By only executing drives which use actuators that are not yet selected the problem in subsumption architectures of inadequate command fusion is solved[47].

Designers of the dialogues use the SpirOps editor to create the drives. Within a drive designers link components to create the output of the drive. These components can be either constants (strings or numbers specific for that drive), or processes (functions processing data). In addition to the standard set of components the SpirOps editor, we added several new ones(see Appendix B). In our implementation of the scenarios each drive contains the program for one topic. The SpirOps editor has an option that allows designers to share drives with each other. As the SpirOps brain executes all drives at the same time this only creates a problem when these drives are used to control the same topic. Our output decision module thus satisfies requirement 4 of the requirements: new dialogues are easy to program and easy to maintain and it is possible to merge the dialogue of two designers easily.

The following list shows the actuators that can be used by the designers, and details what output is required by these actuators:

Speech output A representation of what the robot has to say (see Section 3.6).

Behaviour The name of a program installed on the robot. This could be the name of a gesture installed on the robot.

Topics of which the activation must be changed The designer gives the name of a topic and the new activation.

Handlers to add to a topic A list with names of topics and their new handlers.

Handlers to remove from a topic A list with names of topics and their handlers that should be removed.

Topic of which all heard words should be removed The name of the topic that should be removed.

Memories that need to be set A list with names of the memories and their new value.

Eye colour Colour to be set for the LED in the eye of the robot.

The output decision does not only product speech, but allows for multi-modal output. As described in Section 1.3.1 the focus of our dialogue system was the speech output. WIth this decision module two other modalities are already added: a behaviour and the eye color of the robot. The SpirOps brain allows us to extend to other modalities in the future (for example: adding control for other LEDs, or walking to a position). When a modality is added as an actuator none of the current drives has to be rewritten. This allows us to scale the dialogue system up in the future, when in a follow-up Romeo project more focus is being put on multi-modal output.

3.5.1 Ticket-building

As described in the previous section a declarative method enhances the maintainability of a program. While showing our dialogue design methods to Romeo2 partners designers liked to think in "use-cases" per topic. In topics with several use-cases the motivation and opportunity were the same, but the output was different depending on the situation in which the robot was situated. An example with two use-cases is the topic "medication". The user either took his medication, or did not take his medication. Here two different speech outputs should be selected. If the designer wants to edit this dialogue, this is easy. For example: if he wants to add that Romeo nods his head when the user took his medicine, he adds a behaviour to the ticket (which only takes the addition of one box and one constant). This example can be seen in Figure 3.10. Unfortunately, designers were having problems with creating relevant output per topic in one drive. We solved this problem by adding components to the SpirOps editor that help designers create relevant output for different scenarios in one drive. In each drive designers can create "output tickets". These tickets can contain spoken text, behaviours, and everything else the actuators can handle. A ticket also has a boolean, which indicates whether the ticket is relevant in this situation. Using information of the sensors the designer determines whether the ticket is relevant. After a ticket received all information it is given to a ticket selector. When the drive is selected all options are evaluated, a relevant ticket is selected and the output variables of this ticket are given to the actuators.

In our implementation tickets may currently have the following variables:

```
Boolean isApplicable;
SpeechObject robotSpeechOutput;
String behaviourName;
String nameTopicNewActivation; float newActivation;
HanderArray handlersToAdd;
MemoryArray memoriesToAdd;
```

We hypothesise that the tickets enhance the maintainability of our dialogue system. In the previous section we described that we have case-dependent dialogues. As can be seen in Figure 3.10 it is easy to modify the separate cases the designer created. Each of the tickets can be modified separately, and adding tickets and defining when they can be used is easy. Another way to scale the dialogue system is by adding more variables to a ticket. If designers are missing features these can be added to the tickets, and existing drives will still function faultlessly. In this way newer drives can use all new variables in the tickets, while old drives do not have to be changed.

3.5.2 Conversational strategies

In Subsection 2.1.2 we discussed parts of conversational strategies. What conversational strategies the dialogue system uses is important information, as it defines its convenience to users of the dialogue system. In this subsection we discuss how our dialogue manager uses each of the described conversational strategies.

Grounding The natural language generation component described in Section 3.6.2 attempts to ground the conversation. This component does so by:

- Telling the user what state the robot is in.
- Telling the user what the robot expects the user to say.

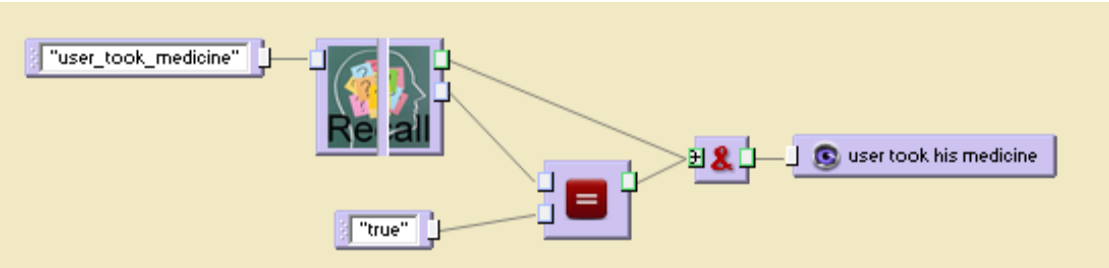
This component does not give too much information (it only tells what the dialogue system understood one time), and does not omit vital details (as it does tell everything it understood once). The current grounding of the dialogue system is not as advanced as the grounding humans use when talking to each other[13]. However, the currently implemented grounding matches the grounding of comparable dialogue systems such as Ravenclaw[9].

Turn-taking The robot talks in these conditions:

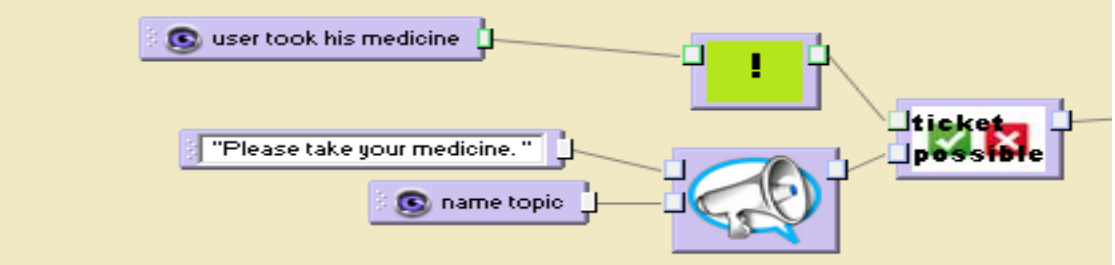
- The user said something (so Romeo will respond)
- A NAOqi event occurred
- The user said nothing for six seconds. During the development of our dialogue system it was helpful that the robot tried to interact with the designer every six seconds. However, when the user is reading a book starting a conversation every six seconds is not a good idea. This condition will be removed when the robot is deployed.

The robot stops talking as soon as the user talks. This creates the possibility for the user to barge-in. The current turn-taking capabilities of our dialogue system are inferior to the turn-taking capabilities other dialogue systems implemented. One example of information other dialogue systems use in turn-taking are nonverbal cues of the user, which benefits the dialogue[17]. Future versions of our dialogue system should thus improve the turn-taking strategy.

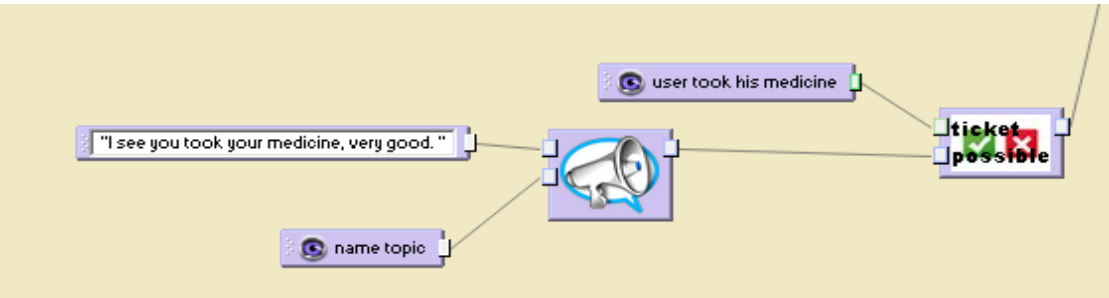
Dialogue mechanisms Dialogue mechanisms can be created by adding drives to the SpirOps brain. Per dialogue mechanism one drive should be created, possibly with a topic per mechanism. To demonstrate this we implemented the "repeat" mechanism. When the user utters "repeat" the repeating topic will get the highest activation. The robot will then repeat what it said, and the activation of the repeating topic will go back to zero.



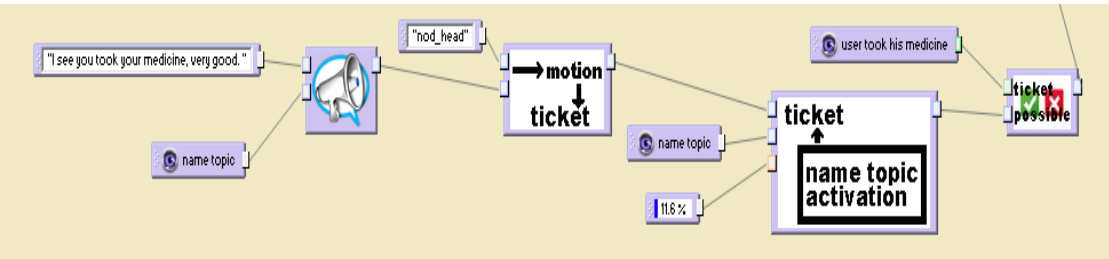
(a) The memory module (biggest box to the left) tries to load the value for the key "user_took_medicine". If it is found (upper line) and the value of this key is equal to true (the lower box tests this) the value "user took his medicine" in the SpirOps drive will be set to true.



(b) The "microphone" box at the bottom receives the name of the topic and the plain text output the user should hear when he did not take his medicine. This box creates a ticket that might be given to the output. If the "user took his medicine" variable is false the generated ticket can be given to the output, this boolean is added in the "ticket possible" box.



(c) Like Subfigure 3.9b a ticket is constructed with output that might be given to the user. In the figure above the ticket can be given if the "user took his medicine" variable is equal to true.



(d) In the above subfigure the ticket from Subfigure 3.9c is extended with more output. The output is still the same, but the motion "nod_head" is added(in the box in the center), and the activation of the topic is reduced to 11.6 percent in the big box on the right.

Figure 3.9: Several parts of Figure 3.10 of the drive "medication". Each part has an explanation so the reader can understand how a SpirOps drive works.

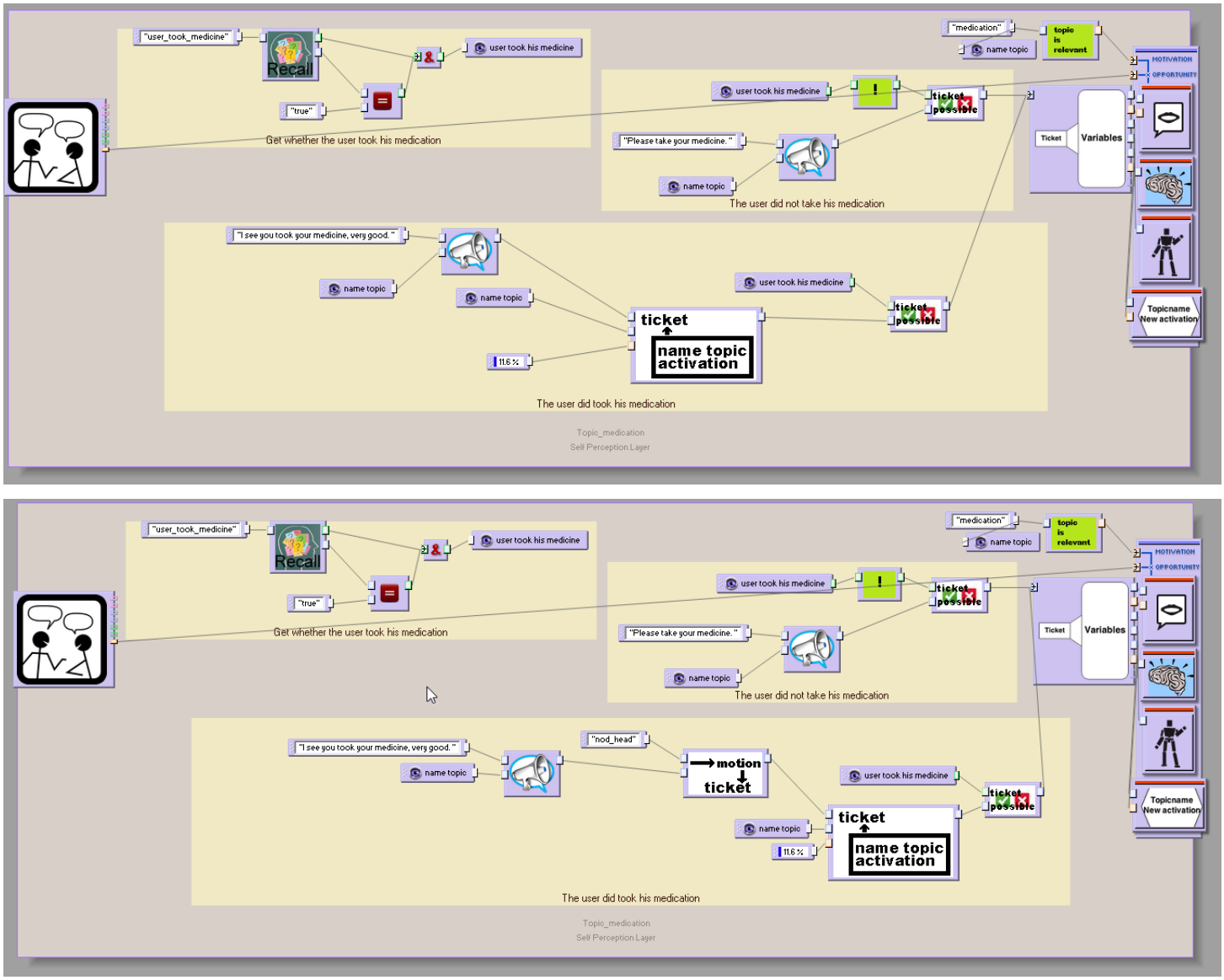


Figure 3.10: Two images of the drive "medication": in the first drive the tickets only give spoken output. In the second drive a behaviour is added to the output. In both drives the activation of the topic is reduced when the user has taken his medication. An explanation of the basic principles of a SpirOps drive can be found in Figure 3.7, an explanation of each of the components surrounded by a yellow box can be found in Figure 3.9. The box where the tickets go in selects a ticket that is applicable for this situation (as indicated in the "ticket possible" boxes).

By adding dialogue mechanisms as drives the complete dialogue system can be improved easily. Adding a dialogue mechanism can be done by any developer, and can be shared with other developers. In this way our dialogue system can improve over time.

3.6 Natural language generation

The natural language generation is responsible for creating understandable texts as output for the user[44]. The language generation component uses information supplied by the designer of the dialogue, and information of the dialogue state. Natural language generation is a big field in computer science, with many existing implementations[44]. Due to time-constraints we did not perform an extensive study on what existing language generator would be the best for our system. Instead, we developed three ad-hoc natural language generators:

Manual specification: the designer writes in plain text what the robot should say.

Output of the dialogue representation: the robot tells the user what the relevant variables of the current dialogue state representation are. It does so by telling what the topic is (if changed), what handlers are not yet satisfied, and what object it found per handler (more information below).

Smalltalk: says something about the world model, only used in the smalltalk drive(see Section 4.2).

As the set-up of our dialogue system is modular it is easy to create a new language generator or add an existing one.

All language generation components have been programmed and designed by us. One of the downsides of all components is that the syntactic variance in the output of the sentences is low. We know that there are better language generating techniques that produce higher quality texts[44]. Future versions of our system should include a more advanced language generator.

3.6.1 Manual specification

The manual specification states both the topic, and the text the designer provided. When a designer wants Romeo to say something specific, this is what he should use.

```
Data: String textOfProgrammer
Data: Topic currentTopic, Topic previousTopic
Result: String outputSentence
outputSentence= "";
if currentTopic.name ≠ previousTopic.name then
    outputSentence = "We are switching from the topic " + previousTopic.name + " to
    the topic " +currentTopic.name ;
end
outputSentence += textOfProgrammer;
return outputSentence
```

Algorithm 1: Algorithm for manual specification of output text

3.6.2 Output of the dialogue representation

By telling the user what state the robot is in the conversation should become grounded (see Section 3.5.2). The dialogue state representation was discussed in Section 3.2. By adding an information state output to the possible natural language generators, designers do not have to write these specific outputs themselves.

The information state output generation module is created with two design principles in mind:

- Only tell new information or changed parameters[35]. It is not necessary to keep reminding the user each time what the topic is.
- Use an implicit confirmation strategy[26, 35]. Information the system heard will be implicitly confirmed by stating the received information. Users can then correct the system by giving the correct information.

Depending on how sure the robot is that the user uttered a certain word, it chooses an output sentence. In the algorithm we use three thresholds for three different outputs. These thresholds can be adjusted by the designers of the dialogue. The current values worked well while we were working on the system, but future research should determine what the optimal values are.

```
Data: Category, BestMatchingWord, Sureness
Result: SentenceToPronounce
SentenceToPronounce= "";
ThresholdUnknown = 0.1;
ThresholdKnown = 0.45;
ThresholdSure = 0.75;
if matchForBestMatchingWord < ThresholdUnknown then
    SentenceToPronounce += " Let me request the " + Category ;
    return SentenceToPronounce;
end
if matchForBestMatchingWord > ThresholdSure then
    SentenceToPronounce += We know what you want for category: " + category;
    return SentenceToPronounce;
end
if matchForBestMatchingWord > ThresholdKnown then
    SentenceToPronounce += " So for category: " + category + " we take " +
    BestMatchingWord;
    return SentenceToPronounce;
end
SentenceToPronounce += " We guess that you want " + BestMatchingWord + " for
category " + category ;
return SentenceToPronounce;
```

Algorithm 2: The algorithm for output given the sureness of the handled word

```
Data: CurrentTopic, PreviousTopic
Result: SentenceToPronounce
Handlers = CurrentTopic.getHandlers();
HandlersLastTime = PreviousTopic.getHandlers();
Sentence= "";
ThresholdUnknown = 0.1;
if CurrentTopic.name ≠ PreviousTopic.name then
    Sentence = "We are switching from the topic " + previousTopic.name + " to the topic
    " + currentTopic.name ;
end
for each Handler in Handlers do
    AllCategories = Handler.getCategoriesHandling();
    for each Category in AllCategories do
        if Handler.satisfaction < ThresholdUnknown then
            Sentence = getSentenceToPronounce(Category, "None", Handler.satisfaction);
        else
            FoundHandler = false;
            if Handler.name in HandlersLastTime then
                HandlerLastTime = handlersLastTime.getFromName(Handler.name);
                FoundHandler = true;
                if not handlerLastTime.activationChanged then
                    continue;
                end
            end
            matchForBestMatchingWord = handler.getSatisfactionForCategory(category);
            bestMatchingWord = handler.getBestMatchingWordForCategory(category);
            Sentence = getSentenceToPronounce(Category,BestMatchingWord,
            matchForBestMatchingWord);
        end
    end
end
return Sentence;
```

Algorithm 3: The algorithm for state output

3.6.3 Smalltalk

The smalltalk output component is only used when the topic "smalltalk" is the most relevant topic. Using the conversation state and world model Romeo tells the user what object the conversation is about (for example: about the user himself, or his dog). Romeo also selects a property of this object and asks what the current value for this property is (for example: is the user happy? Or: how old is the dog?). A detailed description of how we implemented the smalltalk component, and the world representation, can be found in Section 4.2.

If the user gives an answer that is expected for the posed question (for example: "yes" to a question that can only be answered with "yes" or "no") the robot updates the world model with this answer. If the user does not answer the question (which indicates that he does not know what he can answer), the robot says what kind of answer the user should give. How the language output is generated can be seen in Algorithm **??**.

```
Data: CurrentTopic, PreviousTopic
Data: SmalltalkState
Result: String outputSentence
Sentence= "";
if "smalltalk" ≠ previousTopic.name then
    Sentence = "We are switching from the topic " + previousTopic.name + " to the topic
    smalltalk";
end
if SmalltalkState.hasAnswer() then
    Sentence += SmalltalkState.getAnswer();
    return Sentence
end
if previousTopic ≠ SmalltalkState.topic then
    Sentence += "I want to talk about " + SmalltalkState.topic ;
end
Sentence += SmalltalkState.topic + " " + SmalltalkState.property + "?";
if previousNamePropert == SmalltalkState.property AND previousTopic ==
SmalltalkState.topic then
    Sentence += " I am expecting you say something in the category " +
    SmalltalkState.nameHandler ;
end
return Sentence;
```

Algorithm 4: The algorithm for Smalltalk output

3.7 Speech synthesis

The speech synthesis component converts written text to the sound waves that the robot will emit when "speaking". The ALTextToSpeech component (which is installed on Romeo) is able to do this. As this component is easy to use and easy to integrate in our program we chose ALTextToSpeech as our speech synthesis component.

Chapter 4

Implementation details

During the internship the dialogue system was implemented. In this chapter we describe two components that were only referred to in other sections:

A monitoring program: A monitoring tool that helps the designer with testing his dialogues by monitoring what the dialogue state is.

Smalltalk: The component that converses with the user about random subjects.

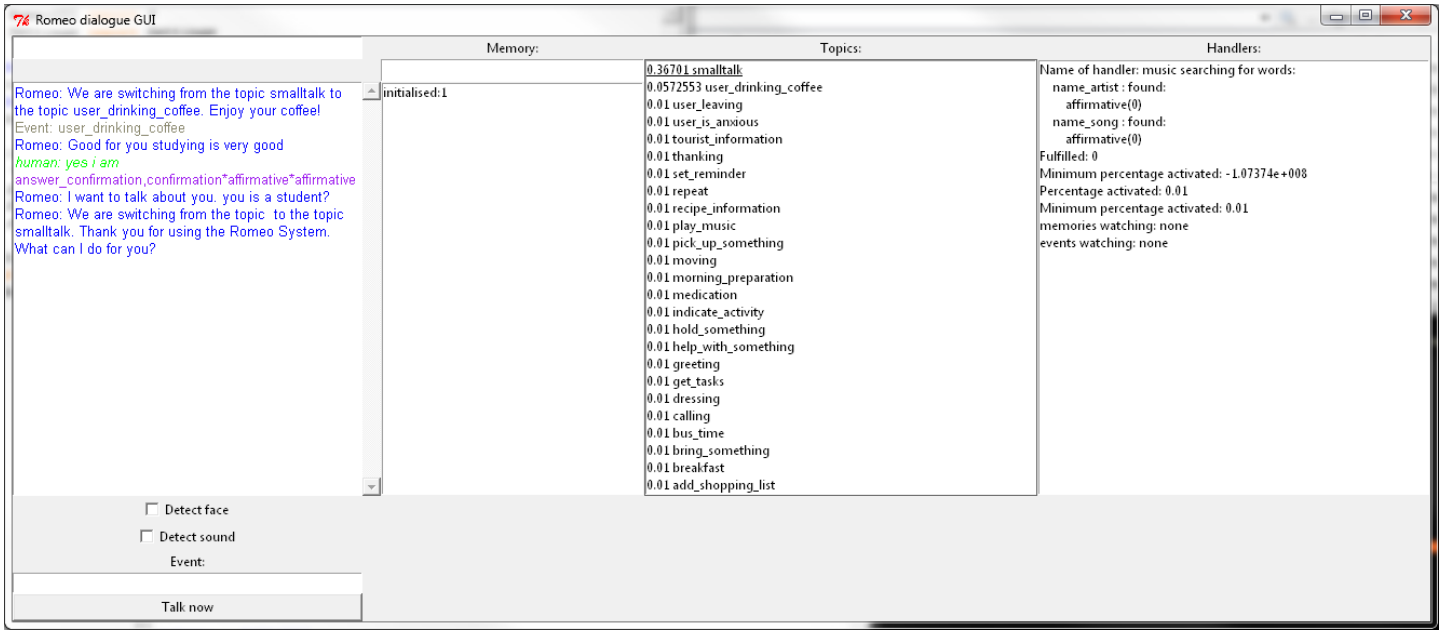


Figure 4.1: A screenshot of the monitoring tool while being used during a conversation. The left column shows the history of the conversation, the middle-left column shows all memories of the robot, the middle-right column shows an overview of the topics including their activation, and the right column shows information about one topic (in this case the topic "play music")

4.1 Monitoring tool

Previous research has shown that making sense of data that is recorded in interaction systems is still a challenge, and that this hinders the development of such interaction systems[36, 57]. Being unable to see why Romeo chose the action it performed, and how the system interpreted the input of the user, makes it hard for the user to improve the dialogue. As a courtesy to the designers a monitoring tool was created to help them with testing and debugging their dialogues. The functionality of NAOqi is integrated in this monitoring tool, which makes it possible to see how the dialogue system reacts to users without having to use a real robot. With this tool a team of designers can work together with only one robot, making our dialogue accessible to design teams with a small budget. The monitoring tool can be seen in Figure 4.1. With the monitoring tool the designer can:

- Read the history of the dialogue. This includes what the speech recognizer recognised, what events fired, and what memories were set.
- Interact with the system by typing, the speech recognition receives the text and can use this with Wit.
- Inspect the memories of the robot.
- See all topics (including activation). When clicking on a topic you see the following properties of this topic:
  - The handlers and what they are trying to catch/handle.
  - The activation of a handler (as described in Section 3.4).
  - What NAOqi memories the topic is subscribed to.
  - What NAOqi events the topic is subscribed to.
  - What the minimum activation is.
  - What the current level of activation is.

4.2 Smalltalk

As described in Section 1.4 one requirement is that Romeo should have a smalltalk component. To demonstrate that it is possible to create a smalltalk component with our dialogue system we created a single drive that handles all smalltalk. When this drive is activated Romeo interacts with the user by asking questions about him and the environment. The world model is used to generate these questions.

As described in Section 1.2 the world model is not yet implemented. However, it is known that it will resemble the ALWorldRepresentation NAOqi component. ALWorldRepresentation stores data in a hierarchical structure of objects and a generic database containing the information on these objects (see Figure 4.3). As ALWorldRepresentation was released after the internship we created an ad-hoc solution during the internship. This solution is a world model which is relational and has a database with information about objects that can occur in the world. This facilitates the substitution of our own developed model by the ALWorldRepresentation component and eventually the world model of the Romeo robot. A timeline of what world model can be used for the generation of smalltalk is shown in Figure 4.2.

Figure 4.2: The timeline of the development of the world model we use to generate smalltalk. During the internship our world model was used, in 2014 ALWorldRepresentation was released which will be substituted by the SpirOps world model in 2015.

The smalltalk component may feed a property that is not yet known into the world model. For example: in Figure 4.2 the name of Human#1 might be unknown. The smalltalk component can now ask "What is your name?". The database with information about objects contains a handler per property (in this example a handler that searches for a name). When the system poses a question about a property it will add the handler to the smalltalk topic. This enables the system to interpret an answer of the user. When the user answers the question in a way the handler understands (for example: "my name is Roland"), the answer ("Roland") will be used to update the world model.

What Romeo is talking about is also available in the world model. As can be seen in Figure 4.3, other components using NAOqi are able to see who the robot is talking to and what it is talking about. This information is used to close the interaction loop between our high-level dialogue component and the sensors of Romeo[37].

The smalltalk component can also add relations to the world model. An example is that the user owns a dog. Once our smalltalk component discovers that this is the case, we add a relation between the user and a dog. This makes it possible to ask questions about this dog. For example; in the generic database dogs have the property "is fluffy". This will lead to the Romeo robot asking: "Is your dog fluffy?".

Other examples of questions the system can ask to the user are:

- "Do you like cooking?", expecting a confirmation or denial.
- "Did you study?" expecting a confirmation or denial.
- "What was the name of your study?" expecting the name of a study.

A small dialogue where Romeo uses the smalltalk component can be found between line 97 and 157 in Appendix C. In Figure 4.4 a visualisation of our ad-hoc world model is shown before the dialogue, as well as after the dialogue. This figure shows that two dogs are added as a relation to our user, and the user's answers are used to fill the unknown information about the user.

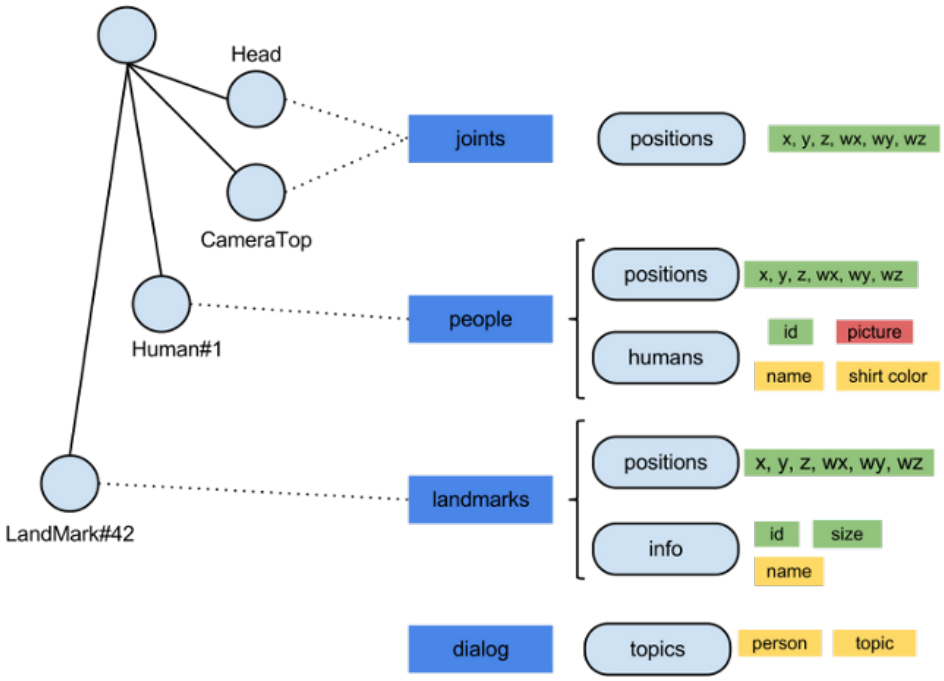


Figure 4.3: The hierarchical world representation for the Romeo robot. The head and camera on top of the robots are represented as joints which have a position. A person in front of him is represented by a person which has a position and all properties humans have. A landmark that can be observed has a position and some information. As can be read in Section 4.2 information about the dialogue is also available in the world model to other components on the NAOqi platform. This image and more information about the world representation can be found on `http://doc.aldebaran.com/2-1/naoqi/core/worldrepresentation.html`

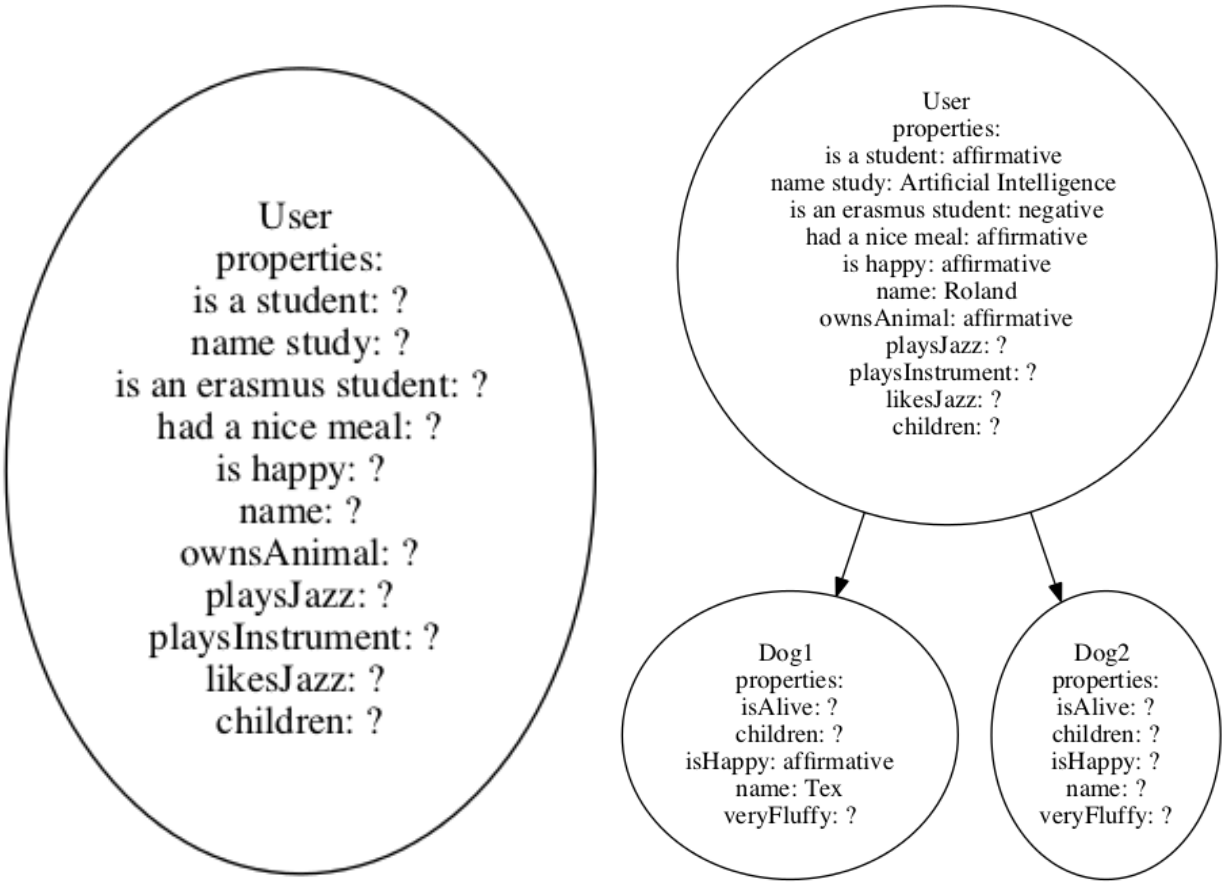


Figure 4.4: The world representation before the user starts to talk (left) and after the dialogue (right). As the user talked with the robot the model was expanded (two dogs are added) and more properties of the user became known.

Chapter 5

Evaluation of the dialogue system

5.1 Introduction

The second research question of our thesis was "How does one evaluate a spoken dialogue system?". When looking at the development of other relevant dialogue systems we found several common approaches. In this section we introduce several evaluation approaches that are relevant to our dialogue system. After we discuss the relevancy of each approach two evaluation methods are chosen. The five relevant evaluation methods are[6, 14, 18, 40]:

Usability benchmark To evaluate the maintainability of the dialogue system a usability benchmark can be developed. In this benchmark the usability of our design software can be compared to other design software, such as Choregraph[6]. For this benchmark participants need to learn how to use each of the components shown in Figure 3.1. The results of the benchmarks will tell us whether our dialogue system is more maintainable than relevant other dialogue systems, and what components in the dialogue architecture should be improved or replaced in a future version. However, training participants to use all components in the dialogue architecture is a time-consuming task[6]. In the dialogue system we focussed primarily on the dialogue manager. We will thus only evaluate the dialogue manager, and do not perform a usability benchmark of the whole dialogue system.

User experiments In Section 1.3 we set the requirements of our dialogue system using the scenarios Romeo will encounter. Testing with the users who will use the dialogue system will tell us whether users behave as expected, and whether Romeo behaves as expected when interacting with hem[14, 26, 42, 58]. It is best to involve users of a dialogue system as early as possible[14], this way improvements of the dialogue system can be made in an early stage. Even though the Romeo2 user experiments were performed before our internship this data was not available during the internship. Due to time constraints no actual experiments were performed.

Expert analysis Evaluating the dialogue system is possible by evaluating each of the components described in Section 3.1. For each of the components a set of usability criteria can be defined[14, 18]. An advantage of this approach is that it shows what components should

be improved. Another advantage is that using experts to evaluate, costs less time than a usability benchmark with multiple non-expert participants. There are also disadvantages to an expert analysis, and Dybkjaer mentions several of them[14]. One disadvantage is that many important issues regarding the usability of a dialogue system cannot be quantified. This lack of quantification makes it difficult to get an objective evaluation by an expert[14], and is the reason an expert analysis was not conducted.

Scenario evaluation As described in Section 1.3.1 there are five scenarios in the requirements document of the Romeo2 project in which Romeo must be able to function. By showing how the dialogue system behaves in these scenarios the reader is able to discover the strengths and weaknesses of the developed system. The errors the system makes during a particular scenario will tell us what the weak aspects of the system are. A scenario evaluation allows us to demonstrate that the novel aspects of our dialogue system work in the scenarios the system was designed for. As both users, experts and designers can see how our dialogue system performs in several scenarios, scenario evaluations are useful for all groups[40].

Functionality benchmark The functionalities of our dialogue manager can be compared to existing dialogue managers. This can be done by testing several qualitative parameters of dialogue managers[14], such as whether the dialogue manager is platform independent and whether it is possible to use topic-unrestricted input. The results of this evaluation are valuable for dialogue designers when deciding what dialogue manager they want to use.

This thesis focused primarily on designing a novel dialogue manager for a dialogue system. Given this focus the most appropriate methods are a scenario evaluation and a functionality benchmark. In the scenario evaluation we observe how Romeo performs in the scenarios for which the dialogue system was designed. In the functionality benchmark we compared our dialogue design software to Choregraphe (see Section 2.2.4), the software that is currently used to design dialogues with the Nao robot. Besides these two tests we discuss whether our dialogue system satisfies all six requirements posed in Section 1.4.

5.2 Scenario evaluation

To demonstrate how the dialogue manager performs we wrote a walkthrough through the scenarios described in Section 1.3.1. All functions Romeo needs to perform in these scenarios are implemented. In this walkthrough the user asks questions to achieve his goals. To prevent possible errors in the speech recognition component (which is not the main focus of our thesis) every utterance of the user in the scenarios is given in written form to the system. The dialogue system reacts on these questions, and sometimes poses new questions. As described in Section 5.1 the Romeo2 user experiments were performed before our internship, and this data was not available during the internship. We therefore decided to construct the sentences the user would utter during the scenario evaluation together with a team of dialogue experts at SpirOps. Sentences the system utters are generated by the dialogue manager. The ad-hoc world model used during the scenario evaluation is not specifically aimed at elderly people. This is the reason Romeo asked whether the user is a student or not. When Romeo will be deployed it will have an improved world model. The reader can find the whole walkthrough of five pages in Appendix C. The layout of the transcript is ad-hoc, and as a courtesy to the reader the transcript is annotated with information about what happens in the dialogue.

In the walkthrough the team of dialogue experts at SpirOps identified errors and improvements. In this context errors and improvements were defined as:

Errors: The system makes an error leading to the user needing to repair the conversation.

Improvements: The user does not have to repair the conversation. However, the output of the robot is less than perfect.

Both help us in the identification of problems, and can be used to define improvements for future versions of the dialogue manager.

The results of the scenario evaluation can be found in Section 6.2.

5.3 Functionality benchmark

In Chapter 2 we examined several relevant existing dialogue systems, and how dialogues are designed with these systems. One of these systems was Choregraphe, the program that is currently used to create dialogues for the Romeo robot. Choregraphe and our dialogue manager are made with different goals in mind, and the choice to use one program to design a dialogue over the other program should be made by the designer of a dialogue. The designer should be able to make a well-considered choice when choosing the dialogue manager he will use to design the dialogues. To aid him with his decision we describe several characteristics of dialogue managers. In Section 6.3 we describe to what extent our dialogue manager and Choregraphe possess these characteristics.

The characteristics have been selected from three papers[8, 41, 52] on the development of dialogue systems that are comparable to our system. The sources of the qualitative characteristics we examine are listed here. A description of each characteristic is given right after the listing of the sources.

- The five characteristics the developers of Choregraphe[41] mention as important: platform independency, event control programming, time-based programming, exploiting all capabilities of the robot and a graphical interface.
- The three characteristics mentioned by Sammut in the paper "Managing context in a conversational agent"[52]: the possibility to reuse existing code, good resource management, and topic-unrestricted input.
- The six characteristics Buhus et al. defined as necessary to effectively support research. These characteristics were identified during the development of Olympus[8]. The characteristics are: reuse of existing code, open-source, transparent, flexible, modular, and scalable.

Platform independency Designers should not have to modify their computer environment to design dialogues. Is the design tool capable of running on any platform (Windows, OS X, and Linux)[41]?

Event control programming Is the dialogue system capable of executing behaviours based on NAOqi events[41]?

Time based programming Is the dialogue system capable of executing behaviours after each other[41]?

Exploit all capabilities of the robot Is it possible to use all actuators and sensors the robot has[41]?

Graphical interface A graphical interface allows anyone to easily implement complex features[41], is there such an interface?

Reuse of existing code Code-reuse lessens the development effort. Designers can focus on combining code in new ways, instead of having to develop everything from scratch[8]. Are there ways to reuse code in such a way that development effort is lessened?

Resource management In what way does the dialogue manager decide what actuators are used[6, 52]?

Topic-unrestricted input Is the input the system understands restricted to one topic? Or is the user able to converse about each topic (hereby switching to a new topic) in the domain the system is made for[52].

Recording and motion generation facilities Is it possible to generate extra motions for the robot, with the same program that is used to design the dialogues[41]?

Open-source The complete source code should be available. This allows researchers to inspect and modify the dialogue system[8].

Transparent Every component should provide detailed accounts of their internal state[8].

Flexible It should be possible to accommodate a wide range of applications. To enhance the flexibility it is important that new technologies can be integrated in the dialogue system. Does the dialogue manager allow the integration of such new technologies[8]?

Modular Functions should be encapsulated in components, as this improves the re-use of code among applications. Are components in the dialogue manager application-independent[8]?

Scalable The system should not compromise scalability for the sake of flexibility or transparency[8]. Is the dialogue easy to extend after it is implemented?

5.4 Maintainability metrics

In Section 1.4 we set maintainability as one of the requirements our dialogue system. This gives rise to the questions if our system is sufficiently simple to maintain when compared to comparable systems. To the best of our knowledge the authors of other dialogue systems do not render quantitative evaluations of the maintainability of their developed dialogue managers.

As a courtesy to future researchers that want to compare their dialogue system to ours, we identified two metrics that we believe are relevant to the maintainability of a dialogue system:

Speech input We would argue that the maintainability of the dialogue is influenced by the length of time that is spent on training the speech input component. As described in Section 3.3 the dialogue system is optimised for Wit as speech input. There are several metrics we can get from Wit:

- The amount of expressions we entered to train the classifier of Wit for our scenarios. The more expressions are entered and annotated the more time is spent by the designer of the dialogue. Therefore, the more expressions are entered the less maintainable a dialogue system is.

- The amount of intents created in our Wit instance. Wit needs to see how the intent is used to recognise it, to do this several training sentences must be annotated. The more intents created the more training sentences are required to let Wit recognise these intents. More intents thus make a dialogue system less maintainable.
- New entities created and built-in entities used. Entities that are already recognised by the Wit parser are for example "Contact" (the name of a person) and "Date/time". The Wit parser does not need any training sentences to recognise these sentences, using these entities does not require any work of the dialogue designer. However, when creating new entities Wit has to learn what the possible words in this entity are and how these entities are used. To train the Wit parser to recognise these entities more training sentences must be supplied by the dialogue designer. Therefore, the more new entities are created the less maintainable our dialogue system is.

Complexity of the drives The maintainability of the dialogue is influenced by the amount of links, constants and processes in the drives (as described in Section 3.5). When a designer has to add more links, constants and processes the drives will take more time to design and become harder to maintain. To the best of our knowledge there is no quantitative test that measures the complexity of the SpirOps drives. We thus listed the amount of links, constants and processes used to create the scenario walkthrough. Additionally we measured the time it took an experienced designer to create a drive with four use-cases with a large amount of links, constants and processes.

Chapter 6

Results from the evaluations

6.1 Requirement satisfaction

Requirement 1: Romeo should understand natural language Users can use single sentences in natural language. As described in Section 3.4 this is sufficient for the Romeo2 project. We conclude that our developed dialogue system satisfies requirement one.

Requirement 2: Mixed Initiative With the dialogue state representation as described in Section 3.2 both the user and Romeo can take the initiative to talk about a topic. Our developed dialogue system thus satisfies requirement two.

Requirement 3: A close mapping between situations and the dialogue The dialogue state representation described in Section 3.2 forces the designer of a dialogue to conceive separate topics the robot can talk about. When designing a part of the dialogue in the SpirOps brain editor there is a close mapping between the situation and the dialogue. This is because the designer must indicate the motivation to talk about a dialogue part dependending on the situation. Our developed dialogue system thus satisfies requirement three.

Requirement 4: Maintainability It is easy to add a new part of the dialogue owing to the dialogue state representation described in Section 3.2 and as well as to the SpirOps brain editor described in Section 3.5. It is possible to merge two separate dialogues as long as they do not address the same topics. Our developed dialogue system thus satisfies requirement four.

Requirement 5: Smalltalk The implementation includes a smalltalk component, described in Section 4.2. Our developed dialogue system thus satisfies requirement five.

Requirement 6: Must fit the SpirOps framework The dialogue system is integrated with the existing SpirOps software. Our developed dialogue system thus satisfies requirement six.

6.2 Results scenario evaluation

In this section we describe our findings with respect to the constructed scenario walkthrough described in Section 5.2. A transcript of the interaction is shown in Appendix C. The transcript contains 177 lines, where Romeo speaks 65 times, and our user speaks 34 times. The other lines describe changing memories, Wit interpretations and comments. As we can see both the user and Romeo were able to reach their goals during the interaction. Together with the team at SpirOps we identified the following errors Romeo made in the transcript:

- Line 64: The speech recognition thinks a shirt can be used as a hat. Before this walkthrough 16 training sentences were used to teach Wit how clothing is used in a sentence. The word shirt had been mentioned one time, and indicating two items in one sentence had been done only one time before. This problem can be resolved by feeding the speech recognition more possible input sentences.
- Line 113: Romeo takes the answer to the calling of Denis, and interprets it as an answer to the question about whether the user is studying. When the user says "yes" this answer is added to the handler of the topic "calling", and to the handler of the topic "smalltalk". In topics where interpreting the answer of the user is important we already designed a verification step before performing an important action. This happens in the "calling" topic on line 106, where the user is asked to verify that he wants to call somebody. Designers of dialogues should keep in mind that this problem may occur.

Some answers by Romeo are not necessarily wrong, but could be improved:

- Line 10 and 18: It would be better if Romeo says "You are welcome" in response to a "thank you". The speech interpreter detects that the user is thanking the robot, but no response has been designed. This response can be added by creating an extra drive.
- Line 24: Romeo should start the music game, but this game is not yet implemented by the Romeo partners.
- Line 48: Romeo says "hello user", it would be better to say something personal to the caregiver. Unfortunately this is not yet possible as the speech recognition currently cannot differentiate the speech of different users. One solution is checking who the person standing in front of the robot is. This can be done with the world model described in Section 4.2, and this error will be resolved in a future implementation.
- Line 162: Romeo keeps talking about the recipe information topic. He switches when the user keeps talking about the new topic. Although the conversation sounds strange, we do believe that Romeo trying to keep the user on topic is not necessarily a bad thing.

One remark we often got during demonstrations was that the grammar Romeo uses is not correct. An example of where this happens is on line 99, where Romeo says: "you is a student?". This problem is caused by the natural language generator. Before Romeo is deployed a better natural language generator should be used.

The activation of all topics during the evaluation run is plotted in Figure 6.1. The moment where two topics are "fighting" to get the highest activation can be seen in Figure 6.2. This figure shows one strength of our dialogue state representation. In the transcript this moment can be found between line 69 and line 79. If the event that occurred was not important Romeo would have ignored the event and the topic would not be changed until the dialogue about indicating an activity would have been finished. You can see that both the user and Romeo can take the

initiative, where Romeo starts talking when something important happens (in this case that the user should take his medication). In the transcript we read that the dialogue system does not forget what Romeo was talking about before it interrupted the conversation. After the user takes his medication Romeo the drive "medication" determines that it is not necessary to talk about medication anymore, and sets its own activation to a lower value. Romeo now restarts talking again about the "indicate activity" topic again. This shows that our approach to create a mixed-initiative dialogue system is working.

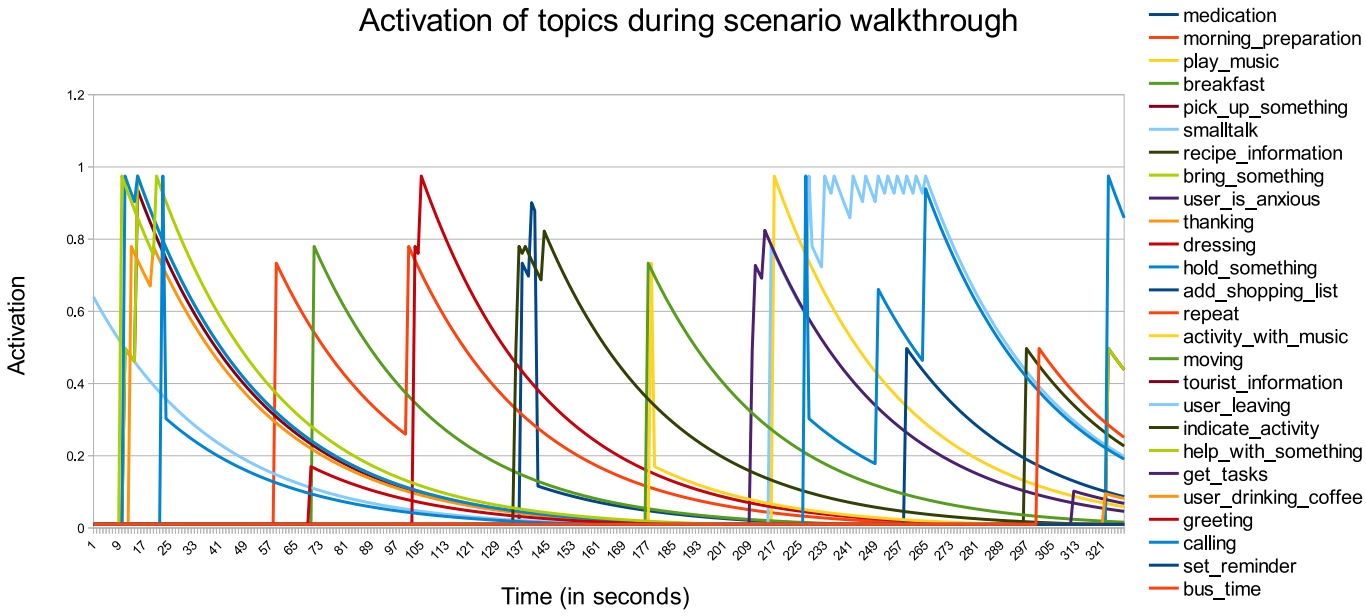


Figure 6.1: The activation of all topics.

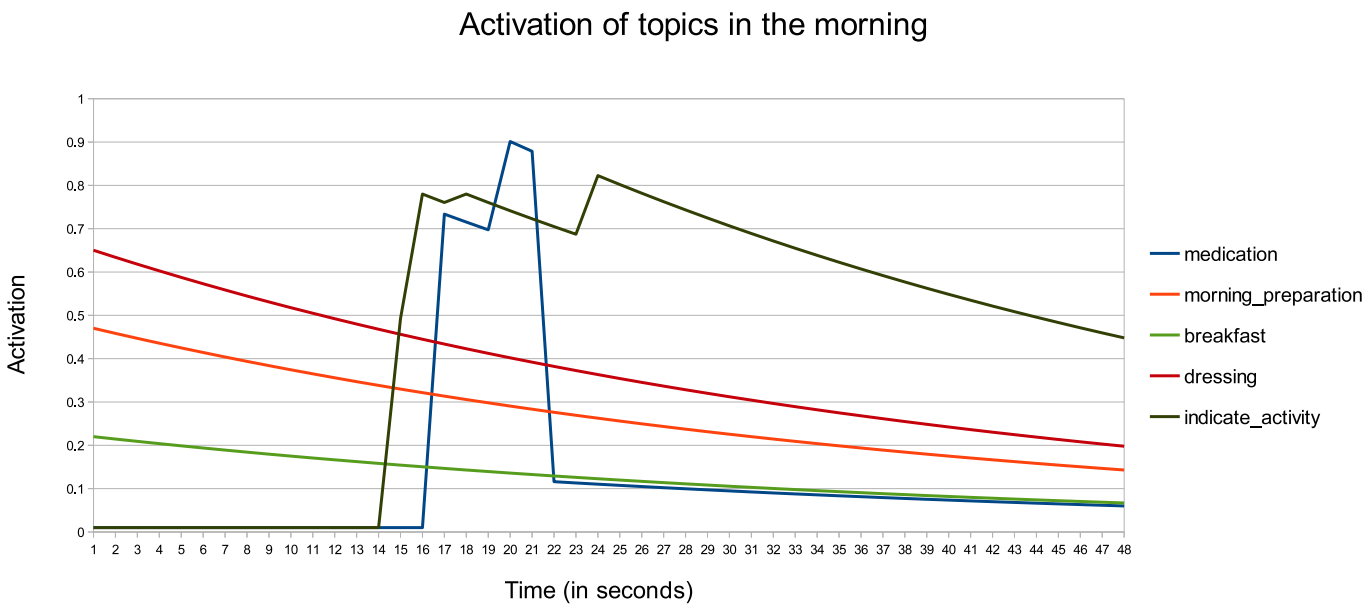


Figure 6.2: The activation of two fighting topics in the morning. At t=14 the user starts talking about what activity he wants to do. At t=17 a NAOqi event fires to indicate that the user has to take his medicine. As the user is talking about what activity he wants to perform he is not warned about taking his medicine. At t=21 a second event fires, which makes the topic medication the highest activated topic, which means Romeo will talk about it. After the user took his medicine the take medication topic sets its own activation to a lower value, and Romeo continues to talk about activities the user can undertake.

6.3 Results functionality benchmark

In Section 5.3 we proposed several characteristics dialogue design tools should have. We compared our system to the Choregraphe tool normally used to design programs for the Nao robot. When a system has the characteristic this is indicated with a green color in the cell, otherwise the cell will get a red color. As described in Section 5.3 a designer should be able to choose a tool based on what he wants to design. In the conclusion we will argue what program is the best tool for the Romeo2 project. Following are the results of the functionality assessment for each program:

Characteristic	Choregraphe	Our system
Platform independency	Capable: it runs on Windows, Os X, and Linux	Incapable: only Windows is supported by the SpirOps editor
Event control programming	Capable: designers indicate what behaviour to execute for each event.	Capable: designers indicate what behaviours to execute per topic. The activation of topics is influenced by events.
Time based programming	Capable: designers indicate the flow of behaviours that should be executed	Capable: designers indicate per topic what should be executed based on the state of the program. The program will be executed when this topic has the highest activation.
Exploit all capabilities of the robot	Possible: there are boxes for each of the actuators and sensors the robot has	Possible: in the current implementation there is no box for each actuator, but these will be added in a future version of the system.
Graphical interface	Yes[41]	Yes
Reuse of existing code	Possible: behaviour boxes can be copied to re-use them.	Possible: SpirOps components can be re-used in different drives.
Resource management	Lock out system. If an actuator is used it will remain locked for other components	Activation system. If a component has a higher activation it will claim this component.
Topic-unrestricted input Text	Incapable: graph based representations (used in Choregraphe) are incapable of dealing with unexpected inputs[52]. Designers enter the set of keywords the system can handle per context.	Capable: input is context-independent.
Recording and motion generation facilities	Capable	Incapable: designers have to use Choregraphe to design the motions and save them on the robot before being able to use them in our program.
Open-source	No. The code is closed source.	No. The code is closed source.
Transparent	Capable: the developer can see the transitions between components including the values they sent while the program is running. Unfortunately there is no log of the interaction available.	Capable: with the monitoring tool described in Section 4.1 the designer can see the internal state of the dialogue manager. The same monitoring tool includes a log of the interaction, including all events on the NAOqi platform.

Flexible	Capable: a wide range of applications is supported. New technologies can be integrated when they are in the NAOqi framework.	Capable: a wide range of applications is supported. New technologies can be integrated as sensors or actuators.
Modular	Capable: all functionality is encapsulated.	Capable: all functionality is encapsulated.
Scalable	Incapable: although it is easy to add functionality to a program, due to the time-based nature of Choregraphe it is hard to add extra topics and the transitions to these topics in larger scenarios.	Capable: drives are unrelated to other drives making it easy to scale.

As can be seen above Choregraphe and our system both satisfy ten out of the thirteen important characteristics for a dialogue system. If it is important for a designer to use a platform independent dialogue system he should choose Choregraphe over our system. However, if it is important for a designer to use a dialogue system with topic-unrestricted input our system is preferable. In the conclusion we will argue that our dialogue system is better for the Romeo2 project.

6.4 Results programmability metrics

In this section we show the obtained programmability metrics.

6.4.1 Speech input

Our Wit instance has:

- 435 validated expressions. Each expression used during the internship was annotated in the inbox of Wit (see Section 3.3.1).
- 31 intents.
- 5 built-in entities. Entities that are created by Wit. Examples are: numbers, datetime and contacts (names of people) that were created by Wit and do not need any training to be used.
- 16 newly added entities. These entities consist of concepts that were not yet created, such as names of dishes and clothing. We had to create these intents for the scenarios, and had to supply training sentences to make Wit understand how these entities are used.

The intent with the most expressions is the "travel_destinations" intent with 100 validated expressions.

To train the speech recognition to be used in the scenario's we had to validate 435 expressions. To the best of our knowledge other dialogue systems did not supply the amount of training they needed to create their dialogue system. New dialogue systems are able to build upon our Wit instance, as it is publicly available. It can be found at `http://wit.ai/cogbot1/Instance+53678f3a`, and can be inspected by everybody who is logged in at the Wit site.

Name drive	Links	Constants	Processes
Topic_bus_time	54	21	18
Topic_dressing	28	9	10
Topic_preparing_at_sink	10	3	3
Topic_medication	36	11	11
Topic_set_reminder	30	10	11
InitialisationDrive	25	13	8
Topic_greeting	13	3	4
Topic_tourist_information	29	10	10
Topic_repeat	17	4	5
Topic_smalltalk	59	15	19
Topic_get_tasks	8	1	3
Topic_indicate_activity	18	9	5
Topic_activity_with_music	13	3	4
Topic_user_is_anxious	45	10	19
Topic_user_leaving	8	2	3
Topic_calling	103	33	30
Topic_play_music	102	36	37
Topic_breakfast	8	2	3
Topic_morning_preparation	102	29	46
Topic_moving	8	2	3
Topic_hold_something	28	4	13
Topic_user_drinking_coffee	8	3	3
Topic_add_shopping_list	42	15	13
Topic_recipe_information	30	10	11

Table 6.2: The drives in the SpirOps brain used to create the scenarios of the Romeo2 project, with the amount of links, constants and processes used to create them (see Section 3.5)

6.4.2 Complexity of creating a drive

To determine what the complexity of drives is we looked at the 24 drives that are currently designed. For each drive we listed the amount of links, constants and processes. The names of the drives including metrics is shown in Table 6.2.

The average amount of links in a drive is 34, the average amount of constants is 10 and the average amount of processes is 12. The maximum amount of links is 103, the maximum amount of constants is 36 and the maximum amount of processes is 46.

To test how much time it takes to design a drive we measured this while programming the "play music" drive. This drive has four cases where Romeo gives different output. There are two handlers that catch the user input for this drive. We as experienced designers created this drive within ten minutes. The drive has 102 links, 36 constants and 37 processes. A video of the creation of the drive can be found at `https://www.youtube.com/watch?v=jmzw3w56G68`.

As can be seen in Table 6.2 the "play music" drive created in ten minutes is the drive with the most links, constants and processes. The dialogue for the Romeo2 scenarios was designed with 24 drives. If each of these 24 drives would be designed in ten minutes the total dialogue of the Romeo2 scenarios would be ready in only four hours. The only dialogue system that shares

information regarding implementation times is Ravenclaw: `http://www.cs.cmu.edu/~dbohus/docs/build_w_ravenclaw.ppt`. Madeleine, operating in the medical diagnosis domain, was created in 22 hours. However, designers using Ravenclaw tell that it actually takes one month to get a working system up and running, and one month to fine-tune performance. Although the dialogue of Madeleine and our dialogue are created for different domains, first results indicate that developing a dialogue with our system is faster than with Ravenclaw.

Chapter 7

Conclusion and discussion

7.1 Conclusion

In this thesis we described the development of the dialogue system for project Romeo2. One unique aspect of this system is the dialogue manager, which performs all tasks a dialogue manager should perform (as described in Section 2.1). The dialogue manager has a novel state representation, and a novel output generation. The dialogue state is represented by topics the robot can talk about, and is updated based on input of the user and NAOqi updates. Designers can independently add topics to the Romeo2 dialogue using the SpirOps editor. As designed dialogues can be merged by sharing the topics of the dialogues with other designers the dialogues are scalable. An important feature is that the dialogue manager updates the dialogue state using information of the components of Romeo2 partners.

Three tests have been conducted to evaluate our designed dialogue system:

- A requirement satisfaction test, where we discussed that our dialogue system satisfies all requirements. These results can be found in Section 6.1.
- A dialogue walkthrough, which showed that the scenarios could be followed with only two errors and five improvements in a transcript of 177 lines. Although our developed dialogue system is promising, there is still room for improvement in the future. The results of the dialogue walkthrough can be found in Section 6.2.
- A functionality check, which showed the strengths and weaknesses of our dialogue system as well as of Choregraphe. The results of the functionality check can be found in Section 6.3.

Based on the first test: the requirement satisfaction test, we can conclude that our system meets all requirements that we set at the start of the project. By satisfying these requirements we can conclude that our dialogue system is suitable for the Romeo2 project. The capability of the dialogue system to function in the scenarios described in the Romeo2 documentation is shown in the second test. The third test compares the existing software to create programs for the Romeo robot with our system. As described in Section 6.3 the two systems have a lot in common. However, there are still important differences that make our system more suitable for the Romeo2 project. Choregraphe is unable to handle topic-unrestricted input, and programs created with Choregraphe are not scalable. As described in Section 1.3.1 scalability is an important factor

for the dialogue system of the Romeo2 project. Two requirements of our dialogue system are that Romeo should understand natural language, and that there should be a mixed initiative topic-unrestricted input. As our system does have these characteristics, and Choregraphe does not, our dialogue system is more suitable for the Romeo2 project. We acknowledge that there are still downsides to our system, as it is not platform independent, does not have recording and motion generation facilities, and is closed source. However, we believe these problems are of lesser importance in the Romeo2 project and solving these problems are challenges for a future version of the system.

The dialogue is represented by dividing the dialogue into topics and giving an activation to each topic. That this approach works can be seen in Figure 6.1, where the activation of all topics during the scenario walkthrough is plotted. With the used update function Romeo can switch to another topic when this topic is more relevant. This can be seen in Figure 6.2, where Romeo switches to a different topic when a more relevant topic gets the highest activation. We conclude that our dialogue representation is a promising approach to dialogue management.

In Section 1.5 we defined the three research questions for this thesis. The first research question was: "what dialogue system has all six requirements described in Section 1.4?". In Section 2.3 we concluded that no existing dialogue architecture is perfect for the Romeo2 project, and thus a new dialogue system should be developed. The second research question was: "how does one evaluate a spoken dialogue system?". In Section 5.1 the five most relevant evaluation methods were listed, including the downsides of each evaluation. We selected the most relevant evaluation methods for the Romeo2 project: a requirement satisfaction test, a scenario implementation, and a functionality benchmark. With these evaluations we tried to answer the third research question: "Is our developed dialogue system usable to design the dialogue of the Romeo2 project?". In this section we concluded that our system is indeed usable for the Romeo2 project.

7.2 Discussion

During the internship we demonstrated our dialogue system to several people, either working at SpirOps or partners in the Romeo2 project. Their reactions were positive, and everybody liked the design method. The fact that they were already using the SpirOps software might have influenced their opinion. The functions of our dialogue system partners liked best were:

- Context-free speech recognition; until now the dialogues designed by partners do not support this feature. All partners were impressed with the fact that we added Wit to the Nao robot.
- The possibility to switch topic by just talking about this other topic. In the current dialogues programmed by them they ask the user to choose a new topic each time a dialogue-tree is finished.

The scenario implementation shows that Romeo is able to cope with all situations in the scenarios. All user utterances were written in a way they fitted in the scenarios. As the data of the Romeo2 user experiments was not yet available for us, we could not establish whether elderly people would use the same utterances. Another question is whether the scenarios are a good reflection of how Romeo will be used. Once this is known the implemented dialogues can be improved and our dialogue system can be re-evaluated.

7.3 Future work

The errors the system made during the scenario evaluation can be attributed to the activation function, the speech recognition, the dialogue state, and the fact that some functions were not yet implemented on Romeo. Several suggestions to improve the dialogue architecture are:

- Change the activation function of handlers (described in Section 3.4). If a topic that is not the most active topic handles a word, the handler should assign a lower score to this word. The system might ask: "X seconds ago I heard you say Y, do you want to do Z with it?". This would solve the problem that occurs at line 113 of Appendix C, where Romeo interprets the answer for a different topic as the answer on a smalltalk question.
- Create a dialogue mechanism that gives the user the option to forget a topic (for example: by saying "forget topic X"). This would give the user more control and solve the problem when Romeo does not talk about the most relevant topic according to the user.
- To create a better grounding between the user and the robot, the robot should indicate so when speech input is not understood. If the activation of topics and handlers did not change after the user gave input, the system did not understand his input. After indicating that Romeo did not understand the user, he can repeat his sentence.
- Currently the designer of a dialogue determines when the system has to remove the words the handlers handled. When the designer does not do this Romeo remembers input forever. This can be a problem for the grounding of the conversation when the user has forgotten that he said something a long time ago. It would be an improvement if the system forgets input after a certain amount of seconds. This can be done by adding a decay to the activation of words captured by the handlers.

One partner had several suggestions to improve the activation of words in a handler. Unfortunately we could not implement these suggestions due to time constraints. We list them here as these suggestions will be implemented in future versions:

- Change the initial activation of a word captured by the handlers based on how likely it is to hear this word. For example: wearing a swimsuit is possible during the winter, but less likely than other clothing.
- Use the probability that the intent is correct, as given by Wit. This function was added to Wit after the internship, which is why it was not implemented in the first version of the dialogue system.
- Use information about previous conversations to create expectations for the input component. If the user often requests an apple when he is hungry it is likely the speech recognition component has to recognise the word "apple" in the context of eating. As described in Section 3.3.1 the parser of Wit is open source and can be edited. This makes it possible to create a new sentence parser for Wit that takes expectations into account. It is also possible to change the activation function of the handlers described in Section 3.4. If the user eats an apple everyday the activation of the word "apple" can be higher when listening what the user wants to eat.

Another suggestion we got while demonstrating our system is assigning an importance to each NAOqi event. Currently events are equally important, but some events might be more important for certain topics. It may even be possible to create an individual activation function for each user. Some elderly should be reminded once each minute to take their medication, while others

can be reminded once every ten minutes. Some users might switch to a new topic easily, while others might not switch very often (for example: people with dementia).

As described in Section 3.6 the syntactic variance of the current language generation components is low. By using an output generation module with templates for each context (such as Rosetta, used by Olympus[10]) the output of the robot can be improved. As the language generation component described in Section 3.6.2 only uses the dialogue state, all topics using this component can be improved without having to change any of the existing drives. Note that a new output generation module does not have to replace the current output generation possibilities of our program. It can be added to components we already created described in Section 3.6. The modal approach of our dialogue system makes it possible to add a better language generator in a later phase, without having to rewrite the existing dialogues.

Work on the Romeo2 project will continue until 2016, and the future work described in this section is already being implemented by SpirOps. With the developed dialogue system the dialogue of the Romeo robot can scale up, and is easy to maintain. Not only speech input of the user used to create a dialogue, but all information on the shared platform can be used[37]. Information about the dialogue is available on the NAOqi platform. This is described by Pandey et al. as the main novelty of the Romeo2 project, as this closes the loop between sensors and the interaction with the user[37]. Finally our dialogue system solves problems regarding the initiative of choosing a topic. Both Romeo and the user can take the initiative to talk about another topic. Our developed system thus is a significant contribution to the Romeo2 project, and eases the introduction of robots in the homes of elderly people.

Bibliography

[1] Choregraphe - Corporate - Aldebaran Robotics — Software: http://www.aldebaran-robotics.com/en/Discover-NAO/Software/choregraphe.html.

[2] SpirOps AI - Scientific Research Lab in Artificial Intelligence: http://www.spirops.com/.

[3] Aldebaran. PROJECT ROMEO - Project Homepage: http://projetromeo.com/index_en.html.

[4] Dimitra Anastasiou, Kristiina Jokinen, and Graham Wilcock. Evaluation of WikiTalkUser Studies of Human-Robot Interaction. *Human-Computer Interaction. Interaction Modalities and Techniques.*, 2013.

[5] John R. Anderson, Michael Matessa, and Christian Lebiere. ACT-R: A Theory of Higher Level Cognition and Its Relation to Visual Attention. *HumanComputer Interaction*, 12(4):439–462, December 1997.

[6] Vincent Berenz and Kenji Suzuki. Usability benchmarks of the Targets-Drives-Means robotic architecture. *International Conference on Humanoid Robots (Humanoids)*, pages 514–519, 2012.

[7] Dan Bohus, Sergio grau Puerto, David Huggins-Daines, Venkatesh Keri, Gopala Krishna, Rohit Kumar, Antoine Raux, and Stefanie Tomko. ConQuest: An open-source dialog system for conferences. *Proceeding NAACL-Short '07 Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, (April):9–12, 2007.

[8] Dan Bohus, Antoine Raux, Thomas K. Harris, Maxine Eskenazi, and Alexander I. Rudnicky. 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*, number April, pages 32–39, 2007.

[9] Dan Bohus and Alexander I Rudnicky. RavenClaw : Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. In *INTERSPEECH*, 2003.

[10] Dan Bohus and Alexander I. Rudnicky. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23(3):332–361, July 2009.

[11] Anne-gwenn Bosser, Guillaume Levieux, Karim Sehaba, Axel Buendia, Vincent Corruble, Guillaume De Fondaumi, and Viviane Gal. Dialogs taking into account experience, emotions and personality. *Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts.*, pages 356–362, 2007.

[12] Herbert H Clark and Edward F Schaefer. Contributing to discourse. *Cognitive science*, 294:259–294, 1989.

[13] HH Clark and SE Brennan. Grounding in communication. *Perspectives on socially shared cognition*, 1991.

[14] Laila Dybkjaer and Niels Ole Bernsen. Usability evaluation in spoken language dialogue systems. *Proceedings of the workshop on Evaluation for Language and Dialogue Systems*, 9:1–10, 2001.

[15] Dirk Fahland, Jan Mendling, and HA Reijers. Declarative versus imperative process modeling languages: the issue of maintainability. *Business Process Management Workshops.*, pages 1–12, 2010.

[16] George Ferguson and James F. Allen. TRIPS: An integrated intelligent problem-solving assistant. *AAAI/IAAI*, 1998.

[17] Terrence Fong, Charles Thorpe, and Charles Baur. Collaboration, dialogue, human-robot interaction. *Robotics Research*, (November), 2003.

[18] Norman M Fraser. The SUNDIAL speech understanding and dialogue project: results and implications for translation. *Aslib proceedings*, 1994.

[19] Rodolphe Gelin, Charles Fattal, Violaine Leynaert, Ioana Ocnarescu, Pain Frédérique, Laurence Devillers, and Mohamed Sehili. Livrable L0.2 Scénarii dusages, 2014.

[20] James Glass. Challenges For Spoken Dialogue Systems. *Proceedings of the 1999 IEEE ASRU Workshop*, 1999.

[21] James Glass and Eugene Weinstein. SpeechBuilder: Facilitating Spoken Dialogue System Development. *Interspeech*, (September):1335–1338, 2001.

[22] Consuelo Granata, Mohammed Chetouani, Adriana Tapus, Philippe Bidaud, and Vincent Dupourqu. Voice and graphical-based interfaces for interaction with a robot dedicated to elderly and people with cognitive disorders. pages 785–790, 2010.

[23] Thomas RG Green and Marian Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing*, pages 131–174, 1996.

[24] Kristiina Jokinen, Topi Hurtig, Kevin Hynnä, Kari Kanto, and Mauri Kaipainen. Self-Organizing Dialogue Management. *Proceedings of the 2nd Workshop on Natural Language Processing and Neural Networks*, pages 77–84, 2001.

[25] Staffan Larsson and David R. Traum. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 6(3&4):323–340, September 2000.

[26] Diane J Litman and Shimei Pan. Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, pages 111–137, 2002.

[27] Seabra L Lopes and António JS Teixeira. Human-robot interaction through spoken language dialogue. *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1:528–534, 2000.

[28] F Mairesse and MA Walker. Towards personality-based user adaptation: psychologically informed stylistic language generation. *User Modeling and User-Adapted Interaction*, pages 1–45, 2010.

[29] Yosuke Matsusaka, Shinya Fujie, and Tetsunori Kobayashi. Modeling of conversational strategy for the robot participating in the group conversation. *INTERSPEECH*, pages 1–4, 2001.

[30] Michael F McTear. Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys (CSUR)*, 34(1):90–169, 2002.

[31] Risto Miikkulainen. Text and Discourse Understanding : The DISCERN system. *A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*, 1999.

[32] Jens-Uwe Möller. Dia-Mole: An unsupervised learning approach to adaptive dialogue models for spoken dialogue systems. 1997.

[33] Tomas Nestorovic. General agent-based architecture for collaborative dialogue management. *2010 2nd International Conference on Software Technology and Engineering*, pages 207–211, October 2010.

[34] Anh Nguyen and Wayne Wobcke. An agent-based approach to dialogue management in personal assistants. *Proceedings of the 10th international conference on Intelligent user interfaces*, page 137, 2005.

[35] Alice H Oh and Alexander I Rudnicky. Stochastic natural language generation for spoken dialog systems. *Computer Speech & Language*, 16(3-4):387–407, July 2002.

[36] E Den Os and Lou Boves. Towards ambient intelligence: Multimodal computers that understand our intentions. *Proc. eChallenges*, 2003.

[37] Amit Kumar Pandey, Rodolphe Gelin, Rachid Alami, Renaud Viry, Axel Buendia, Roland Meertens, Mohamed Chetouani, Laurence Devillers, Marie Tahon, David Filliat, Yves Grenier, Mounira Maazaoui, Abderrahmane Kheddar, Frédéric Lerasle, and Laurent Fitte Duval. Romeo2 Project : Humanoid Robot Assistant and Companion for Everyday Life : I . Situation Assessment for Social Intelligence. *Proceedings of the Second International Workshop on Artificial Intelligence and Cognition (AIC 2014)*, 1315:140–147, 2014.

[38] Joelle Pineau and Geoffrey J. Gordon. POMDP Planning for Robust Robot Control. *Robotics Research*, pages 1–10, 2007.

[39] Joseph Polifroni and Stephanie Seneff. Galaxy-II as an Architecture for Spoken Dialogue Evaluation. *LREC*, 2000.

[40] Alina Pommeranz, Willem-Paul Brinkman, Pascal Wiggers, Joost Broekens, and Catholijn Jonker. Design guidelines for negotiation support systems: an expert perspective using scenarios. *European Conference on Cognitive Ergonomics: Designing Beyond the Product — Understanding Activity and User Experience in Ubiquitous Environments*, 2009.

[41] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. *Robot and Human Interactive Communication*, pages 46–51, 2009.

[42] Antoine Raux, Brian Langner, Alan W. Black, and Maxine Eskenazi. LET'S GO: Improving Spoken Dialog Systems for the Elderly and Non-natives. *Eurospeech*, 2003.

[43] Antoine Raux, Brian Langner, Dan Bohus, Alan W Black, and Maxine Eskenazi. Lets Go Public! Taking a Spoken Dialog System to the Real World. In *Interspeech 2005*, pages 885–888, 2005.

[44] Ehud Reiter and Robert Dale. *Building natural-language generation systems*, volume 1. 1995.

[45] Norbert Reithinger and Elisabeth Maier. Utilizing statistical dialogue act processing in verbmobil. *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 116–121, 1995.

[46] David Reitter and Christian Lebiere. Accountable modeling in ACT-UP, a scalable, rapid-prototyping ACT-R implementation. *Proceedings of the 10th international conference on cognitive modeling, Philadelphia,*, 2010.

[47] J Kenneth Rosenblatt and David W Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. *International Joint Conference on Neural Networks*, pages 317–323 vol.2, 1989.

[48] Nicholas Roy, Gregory Baltus, and Dieter Fox. Towards personal service robots for the elderly. *Workshop on Interactive Robots and Entertainment*, 2000.

[49] Nicholas Roy, Joelle Pineau, and Sebastian Thrun. Spoken dialogue management using probabilistic reasoning. *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 93–100, 2000.

[50] Alexander Rudnicky and Wei Xu. An agenda-based dialog management architecture for spoken language systems. *IEEE Automatic Speech Recognition and Understanding Workshop*, pages 1–337, 1999.

[51] David Sadek, Philippe Bretier, and Franck Panaget. ARTIMIS: Natural dialogue meets rational agency. *IJCAI (2)*, pages 1030–1035, 1997.

[52] Claude Sammut. Managing context in a conversational agent. *Linkoping Electronic Articles in Computer & Information Science*, 3(7), 2001.

[53] Konrad Scheffler and Steve Young. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. *Proceedings of the second international conference on Human Language Technology Research*, pages 12–19, 2002.

[54] Thomas B Sheridan. Eight ultimate challenges. *Robot and Human Communication, 1997. RO-MAN '97. Proceedings., 6th IEEE International Workshop on*, pages 9–14, 1997.

[55] Andrew S Tanenbaum. *Computer Networks 4th Edition*. 2003.

[56] David R. Traum and Staffan Larsson. The information state approach to dialogue management. *Current and New Directions in Discourse and Dialogue*, 2003.

[57] Louis Vuurpijl, Louis ten Bosch, Stéphane Rossignol, Andre Neumann, Norbert Pfleger, and Ralf Engel. Evaluation of multimodal dialog systems. *The Workshop Programme Multimodal Corpora: Models Of Human Behaviour For The Specification And Evaluation Of Multimodal Input And Output Interfaces*, 2004.

[58] Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. PARADISE: A framework for evaluating spoken dialogue agents. *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 271–280, 1997.

[59] Stefan Wermter and Volker Weber. SCREEN : Learning a Flat Syntactic and Semantic Spoken Language Analysis Using Artificial Neural Networks. *Journal of Artificial Intelligence Research*, 6:35–85, 1997.

[60] Stefan Wermter and Volker Weber. SCREEN: Learning a flat syntactic and semantic spoken language analysis using artificial neural networks. *Journal of Artificial Intelligence Research*, 6:35–85, 1997.

[61] Steve Young, Milica Gasic, and Thomson Blaise. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE 101*, 5:1160–1179, 2013.

Appendices

Appendix A

Scenarios of Romeo2

Scenario 1: Preparation in the morning

Every morning the user wakes up and walks through a morning routine. Romeo will guide the user through this routine and helps with tasks that the user is unable to perform.

Scene 1 Romeo helps with washing the user at the sink in the bathroom. He guides the user from waking up to being behind the sink.

- The user is awake, but is still wearing his pyjamas.
- Romeo tells the user it is time to prepare, as the time is 8:25.
- Romeo asks the user to go to the bathroom, possibly Romeo indicates the way to the bathroom.
- The user arrives in the bathroom.
- Romeo invites the user to take place behind the sink.

Scene 2 Romeo helps with washing the user at the sink.

- The user is sitting behind the sink, all objects necessary to wash the user are present.
- Romeo tells the user what tasks he has to perform, and verifies that each task is being performed.
- Romeo tells the user to take his toilet kit.
- Romeo tells the user to use a magnifying mirror.
- Romeo tells the user where he can find a comb.

Scene 3 The caregiver of the user arrives at 9 am. She says hello to the user and to Romeo and proposes to the user to eat breakfast and listen to music.

- A person enters the apartment, it appears to be the caregiver.
- The caregiver says hello to the user and the robot.
- The caregiver proposes to the user to take breakfast.

- The caregiver asks Romeo to play some music.
- Romeo gives a list with possible music, and the user chooses a song.
- Romeo plays the proposed music.
- The caregiver does certain tasks, and Romeo proposes tasks that should be performed.

Scene 4 At 9:30 the caregiver asks Romeo to help the user with finishing the morning preparations and getting dressed.

- The caregiver proposes to continue the morning preparation and to get the user dressed.
- Romeo suggests to go back to the bathroom.
- Romeo continues the morning preparations where they were left before starting with the breakfast.

Scene 5 The user is done with the morning preparations, and the user should get dressed.

- The user moves to the bedroom where the closes is.
- Romeo invites the user to come to the closet.
- Romeo asks the user to choose his top clothes.
- Romeo asks the user to choose his pants.
- The user has chosen his clothing but did not put on his lower clothes. Romeo calls for help.

Scene 6 The caregiver has finished her tasks and proposes to go to the supermarket to buy food for that evening. She will leave the user alone with the robot.

- The caregiver has finished her tasks
- The caregiver is going out to do some grocery shopping.
- The user is alone in the apartment.
- Romeo remembers that the shopping is being done.
- Romeo notes what tasks are finished.

Scenario 2: being alone

Scene 1 The user is alone. This is the moment for Romeo to help the user with things the caregiver proposed.

- All the needs are met for now.
- Romeo asks the user what he is going to do.
- Romeo will propose activities.

Scene 2 Romeo reminds the user to take her medication. Under normal circumstances the user takes her medication before 10:30.

- It is 10:20, the user has not taken her medication
- Romeo tells the user she should take her medication

- Romeo tells how much of each medicine the user should take
- After five minutes, Romeo will check if the user has taken his medication. If this is not the case Romeo will tell the user again that he should take his medication.

Scene 3 When the user is not engaging in any activities during the day Romeo should suggest something to the user.

- It is 10:40 and the user did not engage in an important activity
- Romeo invites the user to call his son, and helps with making the call

Scene 4 When the user starts moving Romeo should ask the user if she wants to use her walker, or if she wants to lean on Romeo while moving.

- The user is moving or is going to move
- Romeo tells the user to use her walker or offers to help with moving.

Scene 5 When the user is not engaged in an activity which requires silence Romeo suggests to put on some music or podcasts.

- The user is not engaged in an activity which requires silence.
- Romeo suggests to put on some music.

Scenario 3: Panic!

Romeo should be able to deal with difficult situations. One of these situations is when the user suddenly gets a panic attack.
Scene 1 The user is worried because he is alone in his home

- The user is anxious
- Romeo asks what the user worries about
- The user tells why he is worried
- Romeo tries to console the user.
- The user is still worried.
- Romeo puts the user in contact with his helper

Appendix B

Components SpirOps editor

As described in Section 3.5 the designer of a dialogue creates drives using the SpirOps editor. We created several new categories in the SpirOps editor with new components designers can use. In this appendix all categories and parts that the programmer can use are listed.

B.0.1 Speech options

Plain text:
input:
name topic: the name of the topic it belongs to (important for the speech output).
speech: what the robot has to say
output:
Ticket that will go to the output.

State topic:
input:
name topic: the name of the topic (which the output generator will use...).
output:
Ticket that will go to the output

Box for a database query:
input:
name database
name request
memory key to set the answer to
output:
ticket

tell answer smalltalk (this is because our program is still a little strange...).
input:
something with a question to ask (container like).

something with our answer
output:
ticket

ask question smalltalk (this is because our program is still a little strange...):
input:
something with a question to ask (container like).
output:
ticket.

B.0.2 Handlers

GetHandlerFromDatabase:
input:
Name handler: the name of the handler you want to get from the database
Surfacename handler: a new name that will be given to this handler (for later reference).
output:
A handler (container).

GetCaughtConcept:
input:
NameTopic: the name of the topic where the handler belongs to
NameHandler: the surfacename of the handler
Category: this is something we still have to remove, i guess...
output:
FoundObject: what the handler found
AnswerFound (boolean): if this handler already found a word.

B.0.3 Ticket chooser

Random possible ticket:
input:
list with possible input tickets: ..
output:
foundOne(boolean): if this handler found a possible ticket.
random possible output: one of the tickets that are possible.

random ticket:
input:
inputTickets: a list with all tickets that may be chosen.
output:
RobotSpeech(string): what the robot has to say
UserCanGiveInputAfterSpoken(float): whether the robot wants to talk after being activated
Memories(struct): a list with memories that will go to our world model

BehaviourName(string): the name of a behaviour the robot has to perform
TopicToReactivate(string): name of a topic which has to change its activation
NewActivationOfTopic(float): the new activation of the topic which had to change
HandlerToAdd(struct): the handlers which have to be added

B.0.4 Ticket functions

AddMemoriesToTicket:
input:
TicketInput(struct): the ticket you want to add them to
Memories(struct): the memories you want to add
output:
ChangedTicket: the ticket

SetConditionBoolean:
input:
ConditionBoolean(boolean): whether this ticket may be used or not
TicketInput(struct): the ticket it is about
output:
ChangedTicket(struct): the output ticket

AddTopicToReactivateToTicket:
input:
TicketInput(struct): the ticket you add it to
nameTopic(string): the topic that should change its activation
NewActivation(float): the new activation
output:
ChangedTicket(struct): the new output ticket

AddMotionToTicket:
input:
Motion(string): the name of the motion you want the robot to perform
TicketInput(struct): the ticket to add to
output:
ChangedTicket (struct): the output ticket

SetWantToSpeakAfterSpokenToTicket:
input:
NewActivation(float): whether the robot wants to speak after it has spoken..
TicketInput(struct): the ticket to add the variable to
output:
ChangedTicket: the ticket with the activation

AddHandlerToTicket:
input:

HandlerToAdd(struct): the handler that must be added
TicketInput(struct): the ticket
NameTopicToAdd(string): the name of the topic that the handler will be added to

B.0.5 Topic Functions

GetActivationTopic:
input:
TopicName(string): the name of the topic that you want to know the activation of
output:
ActivationTopic(float): the activation that this topic has

B.0.6 Memory Functions

MultipleMemories:
input:
Memories(one or multiple) (structs) : the loose memories that you want to combine
output:
MemoryWithArray(struct): a different struct that can be added to a ticket

GetIntVariableFromMemory:
input:
NameOfMemory(string): the name of the memory you want to get
output:
VariableExistsInMemory(boolean): whether this memory actually exists
Vaue(int): the value of the memory
ValueUnequalToZero(boolean): whether the value is unequal to zero (can be used to use ints as booleans)

ForgetMemory:
input:
NameOfMemoryToForget(string): the name of the memory you want to delete
output:
MemoryStruct(struct): what you include in the ticket/multipleMemories box

GetStringVariableFromMemory:
input:
NameOfMemory(string): the name of the memory you want to get
output:
VariableExistsInMemory(boolean): whether this value actually exists in the memory
String(string): the remembered boolean

MakeMemoryStructString:
input:
NameOfMemory(string): the name this memory has
Value(string): the value it will get
output:
MemoryStruct(struct): the memory struct that can be added to the ticket using MultipleMemories

MakeMemoryStructInteger:
input:
NameOfMemory(string): the name this memory has
Value(int): the value it will get
output:
MemoryStruct(struct): the memory struct that can be added to the ticket using MultipleMemories

B.0.7 Smalltalk specific functions

GetAnswerToAnswerSmalltalk:
input:
NameProperty(string): the name of the property the robot is talking about
AnswerUser(string): the answer the user gave (found by the handler).
output:
AnswerToAnswer(string): the answer the robot will give

GetQuestionSmalltalk:
input:
NameHandler(string): the name of the handler we want to add
QuestionProperty(string): the property we are asking a question about (used to memorise and base the answer of the robot on)
QuestionToAskContainer(struct): used to give better output

Appendix C

Transcript

1: NEW STORY! 5.1 scenario intended for elderly people with reduced mobility and fragility
2: Romeo: We are switching from the topic to the topic smalltalk. Thank you for using the Romeo System. What can I do for you?
3: Person:Romeo, can you come?
4: Wit interpretation:help_with_something,task*come?*come?
5: Romeo: We are switching from the topic smalltalk to the topic help_with_something. I will help you with come?
6: Person:Please hold my hand.
7: Wit interpretation:hold_something,object*hand*hand
8: Romeo: We are switching from the topic help_with_something to the topic hold_something. I will hold hand
9: Person:Thank you Romeo
10: Wit interpretation:thanking,
11: Romeo: I will hold hand
12: Person:Can you give me the picture who fell down?
13: Wit interpretation:bring_something,object*the picture who fell down*the picture who fell down
14: Romeo: We are switching from the topic hold_something to the topic bring_something. I will bring you the picture who fell down
15: Romeo: I will bring you the picture who fell down
16: Person:Thank you Romeo
17: Wit interpretation:thanking,
18: Romeo: I will bring you the picture who fell down
19: Person:Can you help me get up?
20: Wit interpretation:help_with_something,task*get up*get up
21: Romeo: We are switching from the topic bring_something to the topic help_with_something. I will help you with get up
22: Person:Can we play a music game?
23: Wit interpretation:play_music_game,
24: Romeo: I will help you with get up
25: Person:I forgot my medicine, can you call my nurse?
26: Wit interpretation:calling,contact*nurse*nurse
27: Romeo: We are switching from the topic help_with_something to the topic calling. Are you

sure you want to call nurse
28: Person:Yes
29: Wit interpretation:answer_confirmation,confirmation*affirmative*affirmative
30: Romeo: I am really really calling nurse
31: NEW STORY! 5.2 scenario intended for elderly people with reduced cognitive functioning
32: A long time passes
33: Start of scenario 1 scene 1 and 2
34: Memory: location_user—bedroom
35: Event: morning_preparation
36: Romeo: We are switching from the topic calling to the topic morning_preparation. we must go to the bathroom for your morning preparation
37: Memory: location_user—bathroom
38: Romeo: Please brush your teeth
39: Memory: brushed_teeth—true
40: Romeo: Please comb your hair
41: Memory: combed_hair—true
42: Romeo: Please fix your make-up
43: Start of scenario 1 scene 3
44: Event: caregiver_arrives
45: Romeo: Please fix your make-up
46: Person:Hello Romeo
47: Wit interpretation:greeting,
48: Romeo: We are switching from the topic morning_preparation to the topic greeting. Hello there my dear user
49: Person:We will now eat breakfast
50: Wit interpretation:breakfast,
51: Romeo: We are switching from the topic greeting to the topic breakfast. We will eat some breakfast. You can always ask me to play some music
52: A short time passes
53: Start of scenario 1 scene 4
54: Person:Please continue the morning preparation.
55: Wit interpretation:morning_preparation,
56: Romeo: We are switching from the topic breakfast to the topic morning_preparation. Please fix your make-up
57: Memory: make_up_done—true
58: Start of scenario 1 scene 5
59: Person:Please help the user to get dressed
60: Wit interpretation:dressing,
61: Romeo: We are switching from the topic morning_preparation to the topic dressing. We should go to the bedroom to pick your clothes
62: Memory: location_user—bedroom
63: Person:I want to wear pants and a shirt
64: Wit interpretation:dressing,lower_clothing*pants*pants-hat*shirt*shirt-upper_clothing*shirt*shirt
65: Romeo: So for category: upper_clothing we take shirt So for category: lower_clothing we take shirt So for category: hat we take shirt
66: Start of scenario 2 scene 1
67: A long time passes
68: Memory: user_activity—none
69: Romeo: We are switching from the topic dressing to the topic indicate_activity. you are able

to indicate an activity. Maybe you want to read a book
70: Start of scenario 2 scene 2
71: Event: take_medicine_1
72: Romeo: We are switching from the topic indicate_activity to the topic medication. Please take your medicine.
73: Event: take_medicine_2
74: Romeo: Please take your medicine.
75: Memory: user_took_medicine—true
76: Romeo: I see you took your medicine, very good.
77: Start of scenario 2 scene 3
78: Memory: user_activity—none
79: Romeo: We are switching from the topic medication to the topic indicate_activity. you are able to indicate an activity. Maybe you want to read a book
80: Romeo: you are able to indicate an activity. Maybe you want to read a book
81: Start of scenario 2 scene 4
82: A long time passes
83: Event: user_moving
84: Romeo: We are switching from the topic indicate_activity to the topic moving. When you are moving your can use your walking stick or use me...
85: Start of scenario 2 scene 5
86: Event: user_activity_can_play_music
87: Romeo: We are switching from the topic moving to the topic activity_with_music. I see you are engaged in an activity where you can listen music. If you want you can tell me what to play.

88: Start of scenario 3
89: A long time passes
90: Memory: user_state—anxious
91: Romeo: We are switching from the topic activity_with_music to the topic user_is_anxious. I see that you are anxious, please try to calm down
92: Memory: user_state—anxious
93: Romeo: I see that you are anxious, please try to calm down
94: Memory: user_state—normal
95: Romeo: We are switching from the topic user_is_anxious to the topic moving. When you are moving your can use your walking stick or use me...
96: Show ability of the user to interrupt a dialogue
97: Person:Let's just talk for a while
98: Wit interpretation:smalltalk,
99: Romeo: We are switching from the topic moving to the topic smalltalk I want to talk about you. you is a student?
100: Person:Can you play a song by Michael Jackson?
101: Wit interpretation:play_music,name_artist*Michael Jackson*Michael Jackson
102: Romeo: We are switching from the topic smalltalk to the topic play_music. I know you want to hear something by Michael Jackson
103: A short time passes
104: Person:Please call Denis for me
105: Wit interpretation:calling,contact*Denis*Denis
106: Romeo: We are switching from the topic play_music to the topic calling. Are you sure you want to call Denis
107: Person:Yes

108: Wit interpretation:answer_confirmation,confirmation*affirmative*affirmative
109: Showing smalltalk
110: Romeo: I am really really calling Denis
111: Person:Go back to smalltalk
112: Wit interpretation:smalltalk,
113: Romeo: Good for you studying is very good
114: Romeo: you name study?
115: Person:Artificial Intelligence
116: Wit interpretation:indicate_study,nameStudy*artificial intelligence*artificial intelligence
117: Romeo: That is a great study as it makes robots like me look smart
118: Romeo: you is an erasmus student?
119: Person:Not at all
120: Wit interpretation:answer_confirmation,confirmation*negative*negative
121: Romeo: Thanks for the answer
122: User ignores robot for a while..
123: Romeo: you had a nice meal?
124: Romeo: you had a nice meal? I am expecting you say something in the category confirmation
125: Person:Yes
126: Wit interpretation:answer_confirmation,confirmation*affirmative*affirmative
127: Romeo: Great that you had a nice meal
128: Romeo: you is happy?
129: Person:Yes
130: Wit interpretation:answer_confirmation,confirmation*affirmative*affirmative
131: Romeo: It is great that you are happy
132: Romeo: you name?
133: Person:my name is Roland
134: Wit interpretation:tell_name,contact*Roland*Roland
135: Romeo: Thank you for indicating the name
136: Romeo: you ownsAnimal?
137: Person:Yes I do
138: Wit interpretation:answer_confirmation,confirmation*affirmative*affirmative
139: Romeo: Animals are great to have
140: Romeo: you ownsDogs?
141: Person:Yes
142: Wit interpretation:answer_confirmation,confirmation*affirmative*affirmative
143: Romeo: Doggies are sooooo cute
144: Romeo: you ownsDogs? I am expecting you say something in the category number
145: Person:Two
146: Wit interpretation:indicate_amount,number*2*2
147: Romeo: Only two you should buy more
148: Romeo: I want to talk about dog of you. dog is happy?
149: Person:Yes
150: Wit interpretation:answer_confirmation,confirmation*affirmative*affirmative
151: Romeo: It is great that you are happy
152: Romeo: dog name?
153: Person:His name is Tex
154: Wit interpretation:tell_name,contact*Tex*Tex
155: Romeo: Thank you for indicating the name

156: Romeo: dog veryFluffy?
157: Showing off other capabilities
158: A long time passes
159: Person:How do I prepare spaghetti?
160: Wit interpretation:recipe_searcher,name_dish*spaghetti*spaghetti
161: Romeo: We are switching from the topic smalltalk to the topic recipe_information. I am searching the recipe for spaghetti
162: Person:I want to take a train from Paris to Amsterdam
163: Wit interpretation:travel_destinations,
164: Romeo: I am searching the recipe for spaghetti
165: Person:I want to depart at 10am
166: Wit interpretation:travel_destinations,datetime*datetime* bladiebla date between Monday 29 September at 10 o'clock and Monday 29 September at 11 o'clock-arrival_or_depart*depart*depart
167: Romeo: We are switching from the topic recipe_information to the topic bus_time So for category: arrival_or_depart we take depart Let me request the location_depart You are encouraged to tell what location_arrival you want So for category: datetime we take bladiebla date between Monday 29 September at 10 o'clock and Monday 29 September at 11 o'clock
168: A short time passes
169: Person:What are my tasks for today?
170: Wit interpretation:get_tasks,
171: Romeo: We are switching from the topic bus_time to the topic get_tasks. I will get your tasks from a database. You still have to do the dishes
172: A short time passes
173: Event: user_drinking_coffee
174: Romeo: We are switching from the topic get_tasks to the topic user_drinking_coffee. Enjoy your coffee!
175: Person:Please hold my mug with coffee
176: Wit interpretation:hold_something,object*mug with coffee*mug with coffee
177: Romeo: We are switching from the topic user_drinking_coffee to the topic hold_something. I will hold mug with coffee