UNIVERSITY OF UTRECHT

MASTER THESIS

Social Agents for Learning in Virtual Environments

Author:
Lucas WEIDEVELD

Supervisor:
Frank DIGNUM

Supervisor:
Manuel GENTILE

Supervisor:
Agnese AUGELLO

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Institute of Information and Computing Sciences
Department of Information and Computing Sciences

June 26, 2017

Declaration of Authorship

I, Lucas WEIDEVELD, declare that this thesis titled, "Social Agents for Learning in Virtual Environments" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Computers are getting smarter all the time. Scientists tell us that soon they will be able to talk to us. (And by 'they', I mean 'computers'. I doubt scientists will ever be able to talk to us.)"

Dave Barry

"I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines."

Claude Shannon

"Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain."

Alan Turing

UNIVERSITY OF UTRECHT

Abstract

Faculty of Science
Department of Information and Computing Sciences

Master of Science

Social Agents for Learning in Virtual Environments

by Lucas WEIDEVELD

It is imperative that health-care professionals are able to communicate well with their patients. Not only verbally, but also in body language. They should show empathy when needed and really know what to say and when to say it. Previous efforts in this work by Jeuring et al. Communicate! - A serious Game for Communication Skills [20] inspire us to expand upon their work. Communicate! gives the ability to expand ones communication skills, but does so in a restrictive manner. The player makes choices from a limited set of responses. In this paper we present our application: SALVE (Social Agents for Learning in Virtual Environments). SALVE is a serious game that helps the player to develop their communication skills, but with less restriction. The player speaks to an agent in a controlled environment, but is free to speak as they wish. The agent responds based on pattern-matching via chat bot technology, assisted by a rule-based interpretation module. Because of the endless possibilities that this combination provides, a social practice is put in place to give a natural boundary to the scope of the system. SALVE offers the abiltiy to have a dialogue with an agent, that shows emotion and accepts empathy. At the end, the player receives a score based on how they did. Development in SALVE is easy because both the pattern-matching and the rule-base languages are close to natural language. In future iterations we hope to improve upon the ease of use of SALVE, as well as the quality of the agents.

Acknowledgements

I would like to acknowledge all the help provided to me by my supervisor Frank Dignum. His inspiration and ideas were always helpful in guiding me to create and improve this work. Also Manuel Gentile and Agnese Augello offered me their support whenever they were able. Both when coding and discussing ideas to further develop the project. I would like to thank my parents and my sister for the continuing mental support, as this project took quite a bit out of me. I would like to thank Jan Willem Stroes for proof reading some of my earlier versions and giving me his thoughts and ideas that helped me improve the thesis. Finally I would like to thank Marco van Os and Betabit for giving me the space to work on my thesis, while still supporting me throughout with good ideas, planning and time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Students that want to become a health-care professional must be able to communicate efficiently and effectively. A good health-care professional should master the skills required to communicate well. However teaching these students all these competences requires lots of training, preferably with trained actors and supervisors. This requires a lot of man power, which is often unavailable at large institutes, where one professor can be expected to teach over two hundred students. To remedy the lack of manpower, a computer program can be utilized. This computer program should however have similar strength in communicating as a human could, at least for that specific topic. Although rivalling the communication strength of a human being is currently still impossible, we must approach it as best we can. In doing so we offer the students and professors of health-care training better tools to learn and teach respectively.

1.2 Inspiration

This project was directly inspired by the work by Jeuring et al. on the Communicate! project [20]. In their work they aspire to improve the training capabilities of interpersonal communication skills between a health-care professional and a patient or client. In this serious game the player (the professional) speaks to a virtual client (the agent) in a scripted dialogue where the player chooses a specific utterance from a predefined list of utterances. The Communicate! game offers a scenario that the player goes through, after which they receive feedback on how they have done. Also, through emotions, immediate feedback can be seen in the game from the agent (patient). Beside the game, they also host a scenario editor that allows teachers to specify their own scenarios, which their students can then use to learn from.

In the handbook of communication and social interaction skills [16], there are multiple criteria that are considered to be a requirement to become competent in communicating. They mention the following:

Clarity - Reflecting an observational world through a natural language. Often typified by statements as "Why can't you just say what you mean" or "Just be clear".

Understanding - Although similar to, and often confused with clarity, it is not the same. Understanding could be defined as the extent at which the speakers comprehend each others' intended meanings. Clarity however is the extent at which the words spoken are 'clear' (i.e. you can be very clear in your message, however not be understood).

Efficiency - A person is efficient at something if he can achieve a given goal with as little as possible resource-intensive, complex or effortful actions. Efficiency is often typified by statements like "Why didn't you just say so (in the first place)?" or "Why are you beating around the bush?"

Satisfaction - A person can achieve satisfaction through interaction because the outcome of the communication is favorable. Satisfaction is often typified by statements such as "I enjoyed talking with you" or "Nice talking to you".

Effectiveness - Effectiveness can be defined as the extent to which preffered outcomes are achieved. Efficiency, satisfaction and effectiveness are closely related. Efficiency defines the effort to reach favorable outcomes, satisfaction speaks of the 'favorability' of the outcome, whereas effectiveness describes how often you reach the favorable outcome.

Appropriateness - Appropriateness is defined as the extent to which behaviour meets the standards of legitimacy or acceptability in a context either by using or altering existing rules or by establishing new rules.

Greene and Burleson mention that the latter two criteria represent "the most general, encompassing, and conceptually useful criteria for competence". Further, they mention that "clarity and understanding are only important to the extent that interactional goals are achieved. Efficiency adds a value judgement that the quickest or least 'expensive' path is always preferable to one that may take more effort but end up being more rewarding".

The intentions of the approach of Jeuring et al. (Communicate!) are admirable, however we believe if we analyze the criteria from the handbook of Greene and Burleson [16], that not all of these criteria are taught or can be taught. This mostly due to the fact that the communication is predetermined and more importantly not self-made. This disallows the student to learn the criteria like clarity (choosing the right words), understanding (the patient's responses are scripted, no understanding is required) and appropriateness (The dialogue is determined by another, thus the student does not learn how to speak appropriately, only that the sentence chosen might be). In contrast to this, we do believe satisfaction can be taught. Satisfaction can be achieved or not in the scenario's based on the choices the student makes. Meaning they determine the outcome themselves: the patient is satisfied with the conversation or not. Whether effectiveness and efficiency can be taught with this system is somewhat vague, as it can be argued that one can become efficient and effective at speaking within the confines of Communicate!, however that does not make them efficient or effective at communicating in a situation where complete freedom of speech is given.

1.3 Research Question

In our project we wish to build upon the approach of Jeuring et al. by allowing more freedom. The Communicate! project is based on a scripted dialogue. In this project we wish to expand on this by allowing the player to respond freely. However, complete freedom is still a goal beyond the scope of this project. The system will be able to handle complete freedom in theory, but the practical side of this freedom would require too much work for a single project. Which leads us to the research question: **Can we create a scenario similar to one in Communicate!, where we allow free communication yet still are able to reach the learning goal for the player?** To test this we will create a prototype that will have a similar scenario to the one in Communicate! however we will make use of different techniques to allow this freedom in communication. We will make use of social practices combined with a rule-based dialogue manager. We expect that more, if not all of the communication criteria of Greene and Burleson can be met using this more open approach. In the future more and more scenarios can be added to the project to be able to study all the different angles of communication.

1.4 Layout

In this paper we will try to answer the research question mentioned above. To do this we must first cover the background information (Chapter 2) about the techniques we will use and we will go more into depth about how Communicate! works. In chapter 3 we will discuss scripted dialogues, as these are used in Communicate!, and we will show some of the advantages and limitations of this approach. The next two chapters (Chapters 4 and 5) will go into the technologies used in SALVE. First, chapter 4 will explain what rule-based dialogues are, as well as lay out its advantages and limitations. After that, chapter 5 will discuss what a social practice is and why it is a good addition to a rule-based dialogue system. In chapter 6, we will go into the more practical side of things by discussing the implementation of the SALVE system. We will lay out the structure of SALVE and then talk about each module in turn. In the following chapter (chapter 7) we compare the two systems - SALVE and Communicate! - to see where improvements have been made and what advantages and limitations both systems have. Finally, we end the work with a conclusion chapter, where we answer the research question and discuss future possibilities for improvement.

Chapter 2

Background

2.1 Introduction

In the course of several decades the field of artificial intelligence has evolved and started taking up a large part of society. In 1950 Turing already asked the question: "Can machines think?" [34]. Only a year after, the first game AI was written for draughts (American checkers) and chess. In 1956 at the Dartmouth Conference, the name Artificial Intelligence was born [22]. As the field grew, more challenging tasks were attempted. Such as taking on the reigning chess champion Kasparov. At first the IBM computer Deep Blue I was not succesful, but after some revisions, Deep Blue 2 prevailed.

In the current society we see AI both supports and challenges people in their daily lives. Think of heating systems that know what temperature your house should be, browsing sites that know what you like to watch. Think of games that have advanced AIs that can take many forms and execute various different strategies to challenge you. Another important development is the use of AI in communication. Natural language is difficult for a computer to learn to understand, but progress is being made in this area every day. One of the ways to handle natural language is via a chat bot. These bots apply pattern matching to respond to what the other person is saying. There are already various upcoming chat bots that will help us in daily life and some even after you have passed [1].

In the next few sections we will outline the theories and technologies we will use, each with its own strengths and weaknesses. We will make use of the strengths and try to combat the weaknesses.

2.2 Chat Bots

To provide the user the ability to speak freely, the agent must be able to reply to whatever the user has to say. A chatbot is one of the forms of AI that manages this task quite well. One of the best ways to establish what a good chatbot is, is with the yearly Loebner prize competition [18]. This prize competition tests chat bots by a Turing test with a jury. The bot that can fool the jury wins the gold medal. If nobody can, the most human-like bot wins. Here we see that not only has A.L.I.C.E. [2] won 3 times herself (2000, 2001, 2004) but also chatbots based on the AIML (Artificial Intelligence Markup Language) [36] technology have won the loebner prize (Mitsuku, 2013). In the article about Mitsuku [3], it even becomes clear that in

the top four, three chat bots used the AIML technology. The ability to create chatbots quickly in AIML, combined with the success of Alice and these other AIML chatbots leads us to believe that AIML is a very interesting way to manage a natural language chat.

AIML is a simple pattern matching language based on XML principles. It is a mark up language with various tags that control the chatbot. The *category* tag represents a question answer module, made up by a *pattern* tag and a *template* tag. The pattern tag encloses the sentence(s) that are compared with the utterance of the user. The template tag encompasses the response the chatbot will give if the pattern is matched. There are various other tags that can be used to expand on this however, such as the *topic* tag, the *that* tag and also wildcards (* and _) which always fire, regardless of the user's input. The topic tag allows the botmaster to cluster categories into a collection. That way these categories only fire when that topic is active. The topic tag also makes those categories more specific. If there are two identical patterns, one within a topic, one without a topic, the category within the topic will be fired before the more general one, if the topic is active. The that tag holds the last sentence made by the chatbot, which allows for a question-answer-answer module. In the example below, you can see an example of the that tag in action:

```
<category>
  <pattern> My name is * </pattern>
  <that> Hello there, what is your name? </that>
  <template> Nice to meet you </star>! </template>
</category>
```

AIML has a lot of advantages. It is easy to understand and very easy to quickly make a small scenario. It is a stateless language however. You can only see as far back as the last sentence, as we can see with the *that* tag. Furthermore, it is nearly impossible to get full coverage of all the things a user could say. Which often leads you to have to write categories such as these:

```
<category>
  <pattern> * </pattern>
  <template> I have no answer for that. </template>
</category>
```

Although this does always give a response, it is not an intelligent one. This is one of the downsides of AIML: providing an intelligent response for every input a human could make, is nigh impossible. However another strong point of AIML comes in the form of symbolic reduction, or recursion. This is implemented by the *srai* tag. This tag always appears in the template tag (the response). However it is treated as an input.

```
<category>
  <pattern> What is * </pattern>
  <template> That is <srai> </star> <srai> </template>
</category>

<category>
  <pattern> symbolic reduction </pattern>
  <template> a form of recursion in AIML </template>
</category>
```

In the example above we see the use of the srai tag. Now if the user were to ask *'what is symbolic reduction'*, the first category would be fired. This would then give the reply *'That is'* and the srai tag would take the rest of the sentence in the * wildcard and look for a match with that. This would fire on the second category in the example, giving the complete sentence: *'That is ' + 'a form of recursion in AIML'*. The final conclusion one could draw from the strengths and weaknesses of AIML is that it is very powerful when discussing a specific topic, but lacks the ability to keep an overview and a structure of the entire conversation. The topic for this project is specific (health-care), allowing AIML to shine. However, the lack of overview and structure needs to be solved.

When looking at chat bots, there are many interesting chatbots that were made using AIML, such as the Humorist bot by Augello et al. [7], which uses AIML to present an agent with a "sense of humour". The Humorist bot is composed of three AIML parts: the set of standard Alice categories (for a normal conversation), the set of humor generation (to tell a joke), the set of humor recognition (to understand a joke). The latter of the three makes use of external resources such as WordNet [21] and CMU [32] pronunciation dictionary to detect the presence of a joke or its features in a sentence.

Another example of a chatbot that extends on the AIML language is SAM (Speech Act Man) [17] by Holtgraves and Han. Besides some minor changes (setting the personality of Sam), they also made some changes to increase the 'humanlike' behaviour of Sam. Four input rules were added to allow Sam to detect common mistakes in chat input: autocorrect (spelling), synonym function, substitution function (for smileys and shorthand notations) and lastly sentence splitters. Furthermore they added a delay speed, to allow Sam to seem like a true human typing his reply. This delay is based on the length of the sentence. Avoiding repetition is another feature to improve Sam's humanlike behaviour. The final goal of Sam is to have a conversation with a human being and analyze their social and cognitive processes.

The improvements that Holtgraves and Han [17] made to Sam also show a couple more weak points of AIML and maybe even chatbots in general. AIML suffers from (in)voluntary typographical and spelling mistakes made by humans. This often results in a weak response like *'I have no answer for that'* when the only problem is one misspelled character or word. Adding the features suggested in this paper could solve this. Although sentence

splitting can be aplied by AIML itself, it is complicated and can lead to confusing responses because of combining responses via the srai tag. Splitting the sentences beforehand could allow for stronger and more easily generated responses. The delay speed is specifically interesting for their application, but is not a flaw of chat bots in general.

2.3 Social Practice

In the previous section on chat bots, we mentioned the weaknesses of chat bots: the lack of structure and overview. To solve this problem, we adopt the Social Practice Theory. A theory developed by amongst others Giddens [15], Schatzki [29, 30], Reckwitz [28] and Bordieu [8]. One of the reasons this theory has gained popularity is the approach they take: instead of treating the individuals or the total picture as a focus point, they treat the actual practice itself as the core unit of analysis.

Translating this theory into the field of Artificial Intelligence, V. Dignum and F. Dignum [12] follow this concept: they mention that social practice is often used as 'yet another aspect that needs to be taken care of in the practical planning'. Instead they choose to put social practice at the heart of the deliberation. We use their definition of social practice, specifically the components they mention a social practice should have:

- *Physical context*, which is subdivided into resources (medical gloves), places (the consultation room) and actors (Bob).
- *social context* like social interpretation (severity of the complaint), roles (doctor, patient, nurse) and norms (being polite)
- *activities* (exchanging small talk, discussing symptoms)
- *plan patterns* (steps to follow)
- *meaning* by which they mean the social effects of steps that are taken (comfort, understanding, anger)
- *competences* (medical skills and knowledge)

We believe these components give a quick, but thorough overview of what a social practice entails.

Another work by Dignum et al. [11] describes an extensive social framework based around social practices. They use some similar components, but also expand upon the previously mentioned framework. One of the extensions is the addition of drives or motives. The motives they mention are *achievement, power, affiliation and avoidance*. The achievement motive is quite simple and already widely (implicitly) used in BDI. This motive is about achieving a goal state. The power motive is about having an impact on the world and having control. The affiliation motive drives people to seek, establish and maintain relationships. And lastly the avoidance motive drives people to avoid conflicts or bad situations. Other components of the social framework are: the identity of the agent and its values, the norms of the

agent, its habits and finally its social practices. In their work, social practices have three elements: *Material*, covering the physical aspects. *Meaning*, refering to the social aspects of a situation and finally *Competence*, referring to the skills and knowledge of the agent. Because of the large similarity between these two works, we base our work on the theory of parts of both frameworks. The definition of social practice is the part that is shared by both works and also the part we use in our work. However the theory described in the work of Dignum et al. [11] is part of the aims for the future (see Future Work section 8.2).

This section gave a very small overview of what social practices entail and where they came from. However we will discuss social practices at length in chapter 5 to give an in-depth view of why this theory is so prominent in our project and how it can overcome the flaws of AIML.

2.4 Communication Training

Communication training is a very important aspect of any education in the medical field. Any health-care professional should be able to communicate effectively with their clients/patients. In the work by Yedidia et al. [37], we see a study showing this. They did a test at three different schools, with an comparison group and an intervention group, with 138 and 155 students respectively. The students were tested at the beginning and end of their third year. The intervention group had a comprehensive communications curricula, whereas the comparison group did not. The intervention group showed improvements in each of the three schools for the values they mention in the paper:

- Relationship development and maintenance
- Organization and time management
- Addressing patient assessment
- Negotiation and shared decision making

Yedidia et al.: "These skills have been associated with such outcomes as:"

- Increased patient satisfaction
- Decreased worry
- Improved retention of information
- More comprehensive medical histories
- Improved patient adherence to treatment plans
- Reduction in symptoms
- Improved physiological outcomes
- Higher functional status
- Reduction in malpractice claims

A slightly more theatrical approach was done by Jacobsen et al. [19]. They agree with the importance of communication training and wish to examine the possible positions and functions of the fourth wall in medical communication training. For those unfamiliar with the fourth wall principle, it is an (often imaginary) wall that is between the actors of a theatre and the audience. The function of the wall is to allow the actors to act more 'natural' and allow the audience to observe the fiction as if they were observing real life. They found three different positions for the wall that all contribute to the learning process.

In both articles we can read that the students were supported by either a moderator [19] or a *standardized patient*, which is essentially an actor trained to be a patient. This shows one of the primary problems of communication training. Each student or sometimes student pair requires a third educated person to train or supervise the student(s).

2.5 Communicate! - A case study

In the previous section we showed the importance of communication training and the effects it can have. We also pointed out one of the major problems with communication training. The article by Jeuring et al. [20], emphasizes this issue as well: "Practicing communication skills is labour intensive and requires quite a bit of organization: students practice in pairs, together with an observer, or a student practices with an actor, again together with an observer". This is why Jeuring et al. have developed Communicate!, a serious game designed to increase the possibilities for practicing interpersonal communication skills between health-care professionals and their patients. This game, in turn, reduces the amount of labour required to give a productive lesson on interpersonal communication.

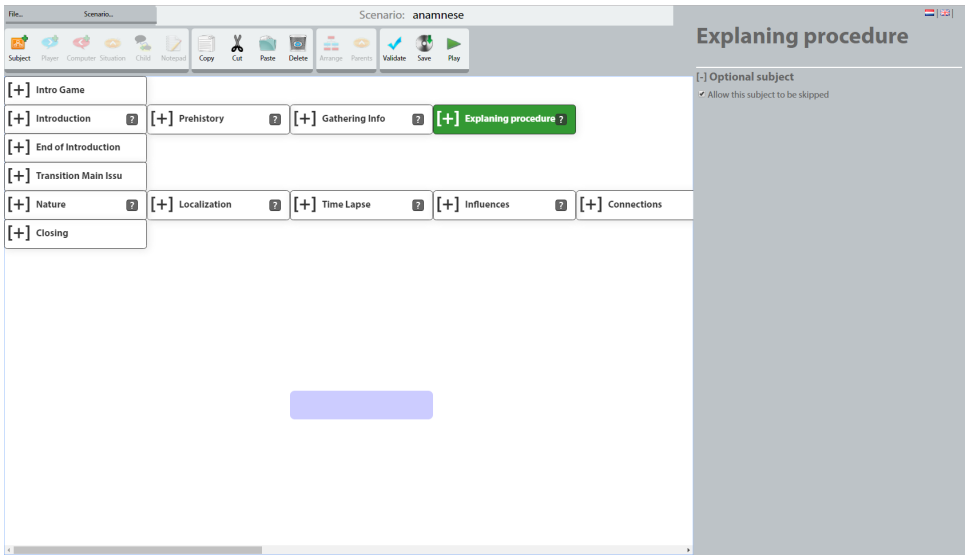

FIGURE 2.1: The Communicate! editor. Structure view

Communicate! makes use of scripted dialogue (see chapter 3) to provide precise scenario's to test their students. The game comes with an editor to create your own scenarios. The instructor can make an entire conversation

in it and specify many things, not just the utterances themselves. Think of personal feedback, emotional responses, changing (scoring) parameters, requirements for an utterance, intentions and comments. The actual interface itself shows a graph like structure (see figure 2.1), to determine the overall structure. Once you enter a specific topic, you can see a tree-like structure, with a start and an end point or multiple end points (see figure 2.2). This allows the instructor to determine multiple paths of going through one topic. Of course it is up to the instructor to determine how many paths to create and which are *correct* or not. If we then select a node, we can define all the aforementioned settings.

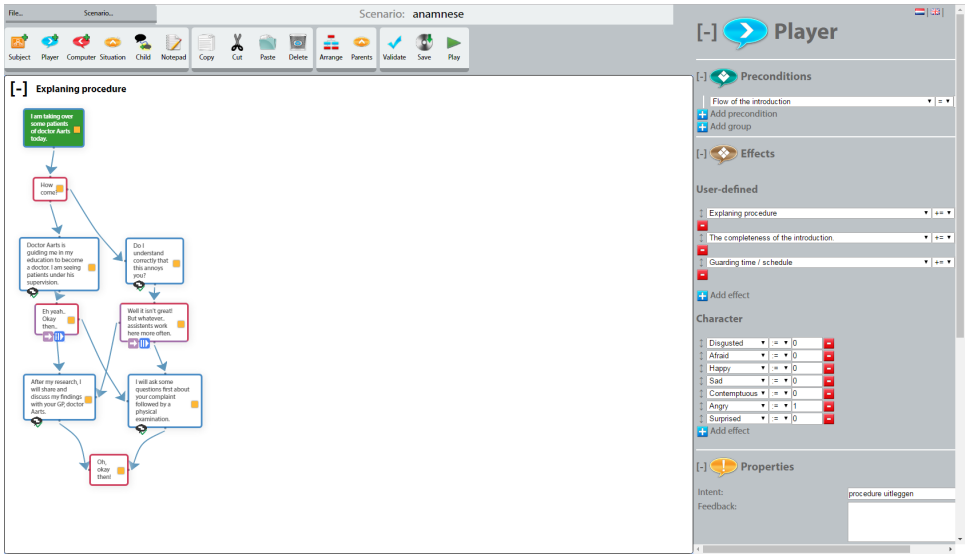

FIGURE 2.2: The Communicate! editor. Tree view

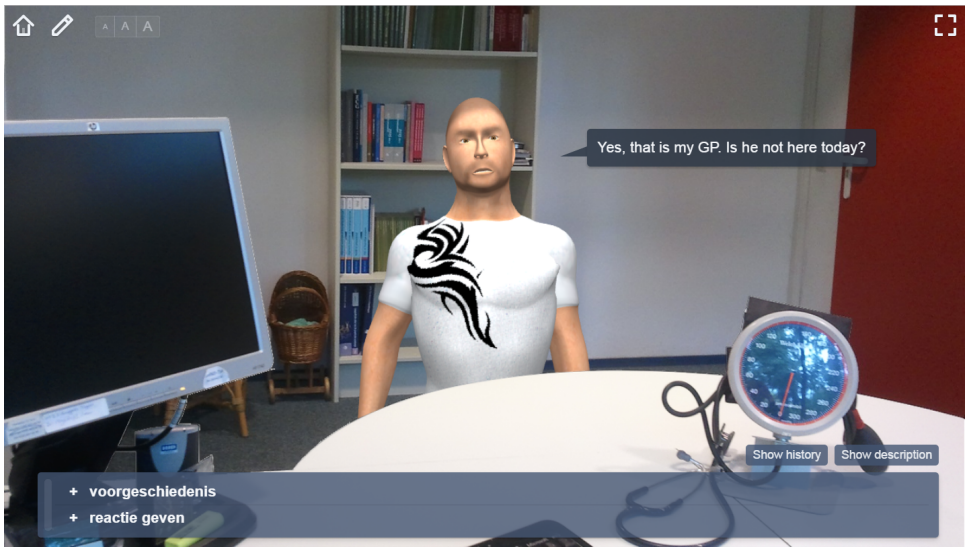

FIGURE 2.3: The Communicate! scenario view.

Besides the editor environment, there is also a scenario environment, where you can play the scenarios you have made (see figure 2.3). This environment shows the patient, the current options, the ability to see the history

of the conversation and the description of the conversation. The patient is also an extra form of information, in the form of their emotional response.

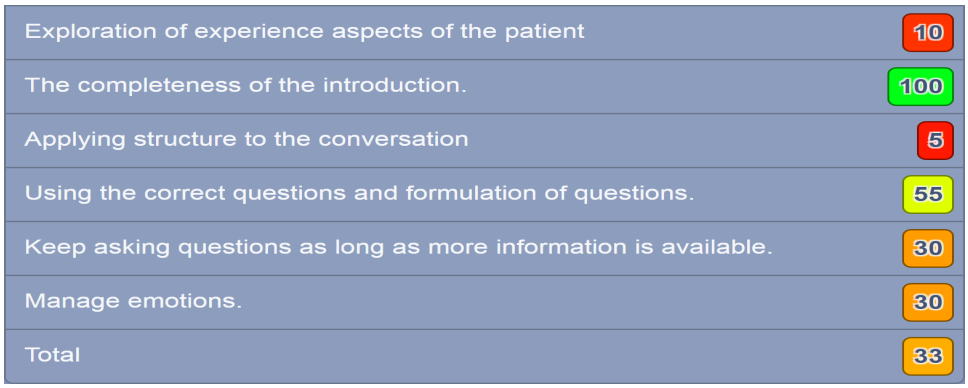

FIGURE 2.4: The Communicate! feedback view (1).

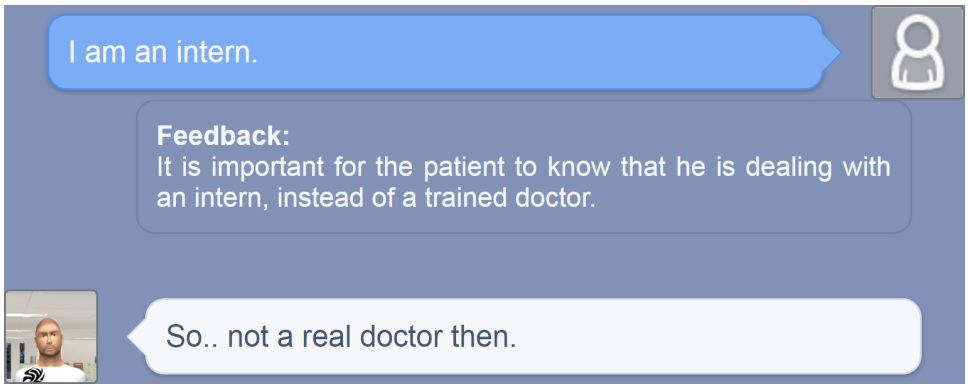

FIGURE 2.5: The Communicate! feedback view (2).

After the conversation is completed, a score screen is given based on the player's performance. Each of the scoring parameters is given with their respective scores in percentages (Figure 2.4). The history is also given with feedback on the choices the player made, provided these were filled in by the instructor (Figure 2.5).

The strength of Communicate! lies in the ability to define nigh endless amounts of conversations very specifically. However, in that strength also lies a weakness: the need to do so. To allow students to train successfully, they should be able to speak as they wish. Nobody will give them options when they are confronted with a patient later. This is what this project will attempt to achieve. The ability to speak (more) freely, and still hold similar conversations as one could define in Communicate!.

Chapter 3

Scripted Dialogues

3.1 Introduction

In this chapter we will discuss the basis of scripted dialogues, as they are used in Communicate!, to make more clear what their advantages and limitations are. First we will give a definition, by means of a small bit of history.

A script, also called screenplay, is a simple written version of a dialogue between multiple players in the game, film, or television program. Often it also contains actions to be taken by the players or actors. Earlier forms of the script appear as early as the 1890s, but are called scenarios [33]. These scenarios not only provided a brief summary but also assisted in marketing and gave helpful explanation to new viewers, as film was a very new phenomenon in that time. The first example of the modern script is the script for the silent film '*A Trip to the Moon*' by George Melies in 1902. The script contained descriptors for the scenes, but also the actions and it provided locations, making it the closest example of a modern screenplay. The person who started using and developing scripts the most in that time was Thomas Ince. He founded Inceville studios and with a large-scale production like that, organization is key. Therefore, Ince required each film to have a script prior to production, which he used to establish the budget required. He also realized that by dividing labour, more efficiency could be realized. This lead to the point where the director was no longer the only one who knew the production plan. Across all parts of the production, every member now had the precise plan of how the movie would be filmed.

However, through all of these changes, sound was still not present. In 1927, the first feature length talking picture was '*The Jazz Singer*'. As sound emerged, the decline of silent films was sure to follow. The dialogue became a major part of a film, and thus of the script. The history of the script goes on to Hollywood and further (See [33]), but this is where we have found our definition. A predetermined dialogue between actors or in this field, agent(s) and actor(s).

3.2 Application

Scripted dialogues are often made by one person and often include not only a message from one actor of the dialogue to the other, but also a message from the scriptwriter(s) to the audience. One of the primary examples of this is commercial advertisement. Here, the actors often have a dialogue with each other about some kind of situation that encourages the use of a

product or service that is offered by a company. In films, the script often also contains information about the rest of the scene, like the actions that should be performed, the location and more. Scripted dialogues in games treat the player as both a participant (Having a conversation with an agent) and as an audience (think of a cinematic where the player does not control the dialogue).

Furthermore, in games, the scripted dialogue where the player participates often appears in code, where the player chooses from a given list of utterances, which leads to an automated response from the agent(s) in the game. The list of utterances is also often tied to a *type of person* you want to be, for example a kind response or a cruel one (see the Fable series [14]). The choices you make also have an effect on the storyline. The choices are all scripted too of course, which provides the game with some replayability.

In conclusion, scripted dialogue is mostly used to tell a specific story or give a specific message, either to a participant in the dialogue, to an audience of the dialogue or to both.

3.3 Advantages

One of the major advantages of scripted dialogues, and scripts in general is the possibility to control the content directly and completely. This also allows the (intended) effect of the script to be clear before the words are spoken. This links to the specialization that scripted dialogues offer. In the work by Piwek and van Deemter [26] we see they mention several advantages of scripted dialogue (generation). One advantage or property of scripted dialogues they mention is the extra layer of communication: the communication between the script writer and the audience of the dialogue. This is the property most used in some of the applications of scripted dialogue as discussed in section 3.2.

Furthermore, Piwek and van Deemter mention that scripted dialogues do not require potentially complicated and error-prone interpretation of the dialogue acts produced by others (remember the work by Holtgraves and Han [17] from chapter 2). They argue that more information is available in scripted dialogue due to the nature of it not being a real-time endeavor. In real-time, one must use the information that was given previously. However, in scripted dialogue, one can have the overview of both preceding and following actions and utterances. Furthermore, they mention it is easier to create dialogues with certain global properties, such as a certain pattern of turn-taking. These properties must otherwise be created individually by the participants, which makes it difficult to control directly.

When looking at the criteria mentioned in the book by Greene and Burleson [16] (see Chapter 1 for definitions), we believe that in a linear conversation, none of the competences can be taught, as the conversation is merely a set of lines that is recited. With the addition of choice, we believe that satisfaction can be taught, as the conversation can lead to a different outcome based on the choices made (for example a good and evil, a positive and a

negative). We believe that effectiveness and efficiency can be taught in the confines of the dialogue itself: one can become efficient at communicating with the given choices, but this does not make him efficient at communicating with unlimited choices. Likewise, one can become effective at communicating with the limited choices, but this does not necessarily apply when they must choose their own words.

It can be argued that for a very specific goal, scripted dialogue is faster than other methods and allows a more specialized message to be sent with the dialogue.

3.4 Limitations

A limitation of scripted dialogue is the necessity to predetermine the entire conversation. This makes it impossible to have a real-time conversation with scripted dialogue. In scripted dialogues the amount of conversation choices is often one (a linear conversation), but as we saw in the application section, scripted dialogues are also used in games where there are multiple scripted responses to a chosen utterance from the player (a dialogue tree). This shows one of the limitations of scripted dialogue: the freedom of the participants is limited by the amount of choices they have per utterance. This freedom, even when there are a near endless amount of choices is limited by the writer of the script, rather than by the intellect of the speaker. This ties to another very important limitation of scripted dialogue, that is underlined by the work of Clark [10], namely the creation of dialogue. According to Clark, the creation of dialogue is an essential part of having a dialogue. The creation in scripted dialogues is managed by the scriptwriter, making it a very big limitation for the participants. He mentions the link between joint activities and language. One cannot participate in a joint activity without language. These joint activities have five dimensions of variation: scriptedness, formality, verbalness, cooperativeness and governance. He mentions that most joint activaties do not come scripted like for example a marriage ceremony. They emerge in time as two or more people try to achieve their goals. This is where the limitation of scripted dialogues shows itself. The goals of the participants are not their own, limiting the dimenions of the dialogue, as all five dimensions are fully scripted and predefined before the dialogue takes place.

Reflecting on the criteria from Greene and Burleson [16], there are various criteria that scripted dialogue can not teach: appropriateness, which requires freedom of speech to learn, as it requires one to choose their own behaviour and language to learn whether these are in line with standards or if new rules can be made to support their behaviour. Clarity and understanding are both criteria that require freedom of speech to teach. They both require learning how to phrase the words to best transfer the meaning you wish to convey. As mentioned in the advantages, the two criteria, efficiency and effectiveness, can be taught in the context of the scripted dialogue itself when choice is involved, however this does not prepare one for the endless choices of phrasing of a (plain) dialogue.

One could argue that the time required for making a script is short, however you can never grasp the same amount of conversation freedom as one could achieve with other methods, such as rule-based dialogues (see chapter 4). Also once a scene reaches a certain number of possibilities, one can lose order in the size of the tree. For example in figure 3.1, we can see a scene where multiple paths are possible. In the end going through the actual conversation once only takes a small amount of time, however the tree that describes the conversation is quite large. So large that reading the text and following the connections in the diagram is impossible.

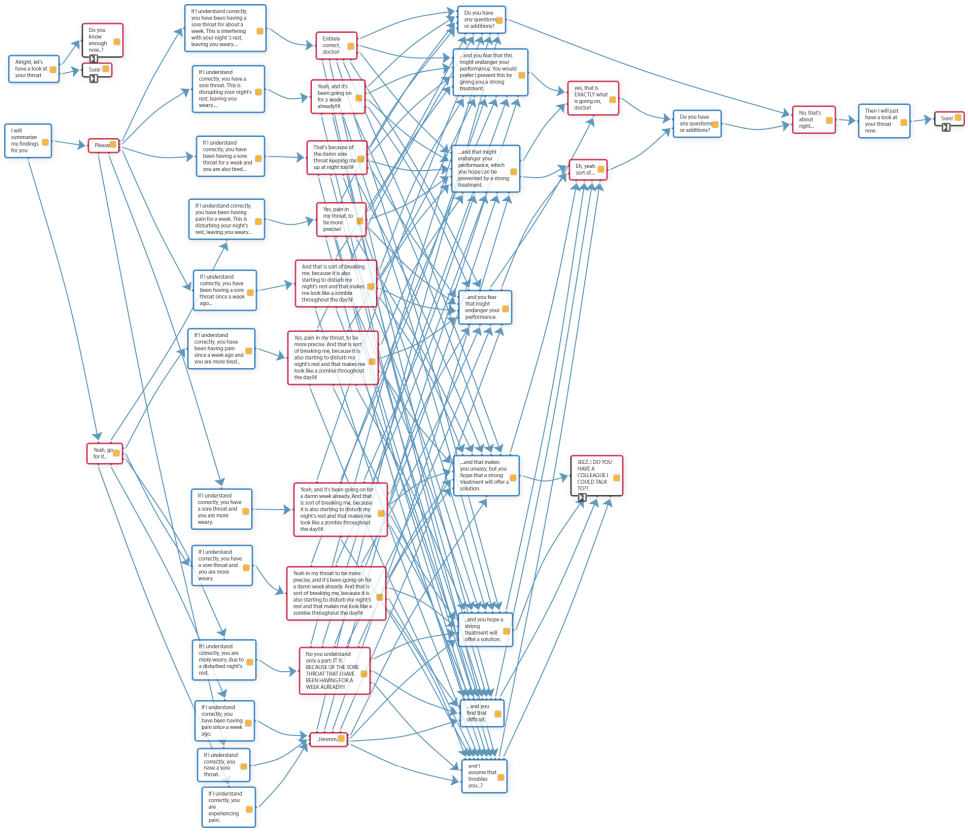

FIGURE 3.1: A large scene from a conversation in Communicate!

Chapter 4

Rule-based Dialogue

4.1 Definition

Rule-based dialogues are dialogues where the participating agents are governed by a rule set or rule base. A rule base is often combined with a memory base or knowledge base which encompasses what the agent knows. Moreover, one must have some kind of interface to communicate with the agent (if human beings are involved). To have a meaningful conversation, the agent must also have a language interpreting module or semantic reasoner of some kind that establishes what has been said. This is turned into actions or knowledge that can be matched against the rule base. The rules then establish the outcome or output of the system. A simple example of how this could work is the addition of knowledge. Say a player mentions something to an agent that the agent is not aware of. The semantic reasoner then interprets what is being said. A rule determines that this knowledge is compared to the knowledge base to see if this knowledge was already in the knowledge base. If the knowledge is not in the knowledge base, then it is added and a response is then given to the sentence determined by the rules that are triggered from this new knowledge. The latter part also happens if the knowledge was already known. In both cases, a sentence can be added to the reply to show the player whether the information was new or already known.

An example of a rule-based dialogue system is TRIPS [5, 6]. TRIPS has an agent-based component architecture, with three main components: interpretation, generation and behavior. It also has two 'smaller' components: the discourse context and the task manager. The discourse context contains information to coordinate the conversational behaviour, such as turn-taking, a model of participators of the conversation, the structure and interpretation of the previous utterance, a discourse history, but also outstanding discourse obligations. An example of a discourse obligation could arise if the system needs to inform the user of something important, right as they have asked a question. The obligation would be that after the important situation has passed, that the system should still answer the question.

The three main components use general problem solving models. These models are formalized as a set of actions that can be performed on problem solving objects. These objects include *objectives* (goals), *solutions* (sets of actions to reach an objective), *resources* (objects used in a solution) and *situations* (settings in which solutions are used to reach a objective). In this model, several actions can be used by agents: *create* (e.g. a new situation,

objective, solution or resource), *select* (e.g. a specific objective to focus on), *evaluate* (e.g. how long a solution might take to implement), *compare* (e.g. two solutions to find the faster one), *modify* (e.g. change a solution to improve it), *repair* (e.g. fix an old solution to make it work) and *abandon* (e.g. give up on an objective). To communicate between agents, Allen et al. describe the following communication acts: *describe* (e.g. describe a solution), *explain* (e.g. explain why a solution works) and *identify* (e.g. select a goal to work on).

The task manager assists in the recognition of what the user is doing, but also the execution of steps to progress on the task at hand. The task manager is able to:

- answer queries about objects and whether they can play a role in the current task/domain
- provide the interface between the behavior agent and the task-specific agents that perform the tasks.
- Provide intention recognition services to the interpretation manager.

The interpretation manager is one of the three main components of TRIPS and is responsible for interpreting incoming parsed utterances and generating updates to the discourse context. The main task of the interpretation manager is to identify the intended speech act, the problem solving act that it furthers and the system's obligations arising from the interaction.

The second main component, the behaviorial agent is responsible for the overall problem solving behaviour of the system. The behavior of the agent is comprised of three aspects:

- The interpretation of the user utterances and actions (produced by the interpretation manager).
- The persistent goals and obligations of the system.
- External events of which the agent becomes aware.

The behavioral agent operates by reacting to incoming events and managing its persistent goals and obligations. When a user-initiated problem is given, the agent determines whether to be cooperative and how much initiative to take to try and solve the problem. This can be setting up an obligation to solve the problem, or even ask the task manager for a (partial) solution and propose this solution to the user. The agent also receives event updates from the world and can choose to inform the user of these. He can choose to simply inform the user or to set up an action to solve the problem.

The generation manager, also a main component, receives problem solving goals from the behaviorial agent and discourse obligations from the discourse context. The task of the generation manager is to synthesize these input sources and produce plans for the system's discourse contributions. When a discourse act or set of acts has been constructed, the generation manager sends these act(s) to the response planner. The planner produces

speech acts and propositional content. When a discourse act is realized and produced succesfully, the generation manager and the discourse context are updated.

TRIPS is already quite advanced in what it can do and has had a lot of time spent making it. Although TRIPS is different from what we are trying to accomplish, it does have some similarities. Agents have their individual intentions (objectives) and act upon these. They make decisions based on the events that have happened and the information they gather from these events. This makes rule-based approaches strong in a social context where these events and knowledge updates are very frequent. This makes rule-based approaches very suitable for the goals we have and therefore we chose to use a rule-based approach in our implementation (see chapter 6).

4.2 Advantages

Returning to one of the limitations mentioned in chapter 3: "Clark [10] mentions that most joint activities do not come scripted like for example a marriage ceremony. They emerge in time as two or more people try to achieve their goals". This, interestingly, covers a big advantage of rule-based dialogue. Rules are made to reply to events that have happened and take action on them as we will see in the structure of rules later on in chapter 6. This approach to events allows rule-based approaches to work very well with the non-scriptedness of general conversations, where each sentence, each action is its own event with its own response.

One of the major points of dialogues is the large amount of information that is exchanged. Clark [10] mentions that one can not enter into a joint activity without the use of language. Even in cases where language is used sparingly, think of a football match for example, information is exchanged through body language or otherwise. Exchanging information is a part of joint activities. Information exchange is one of the advantages of rule-based approaches, as they keep a very up-to-date knowledge base that allows this information exchange to take place fluently.

As we have done before with scripted dialogues (see chapter 3), we will once again review the criteria by Greene and Burleson [16]. The criteria mentioned in the handbook could all be satisfied by a rule-based dialogue approach, provided the approach is extensive enough. Teaching *clarity* requires the player to be allowed to speak in many different ways. Saying the same thing, but in different ways. As the player is allowed to speak in many different ways, the agent programmed with the rule-based approach must be able to exhibit *understanding* for all these various ways. The player can then learn understanding by the agent displaying all sort of responses, to which the player must reply. To be able to reply to the agent, they must understand them. *Efficiency*, *satisfaction* and *effectiveness* all require training to master. Training with an agent with the rule-based approach allows this, as one can have any outcome that is part of the possibilities. This allows one to reach the most desired outcome (Satisfaction), reach this desired outcome with as little resources as possible (Efficiency) and reach it

as often as possible over multiple attempts (Effectiveness). That leaves us with the remaining criterium, *Appropriateness*, which might be one of the hardest criteria to achieve when dealing with agents. Humans often have these rules and standards baked into them over their years of development as a human being (often these traits are missing in small children, but even this is widely accepted to be the case). For an agent to know these same norms and values would require a lot of time and effort to program, but it is not impossible. Vanhee et al. [35] actually wrote about the complexity of adding norms to an agent. They also try to answer questions about how one could implement these norms, how rigid they should be and answer various other interesting questions. They do not go as far as implementing such an agent, but describe designs and ideas as to what needs to be taken into account. From their work it shows that the complexity and amount of code required to fully implement norms in an agent is quite large.

4.3 Limitations

As mentioned in the advantages section above, the criterium *appropriateness* from the handbook of Greene and Burleson [16] is hard to achieve. This makes it a slight limitation of rule-based approaches. Although one could argue that learning 'appropriateness' is hard to do in any approach. Negnevitsky [23] also mentions a similar limitation of rule-based systems: they can not learn; rules do not modify themselves, they do not know when to "break the rule". This is one aspect of humans, that is often found in dialogue: breaking a rule or norm. This is actually a vital part of learning appropriateness and must be learnt.

One of the limitations of rule-based approaches is the amount of time required to set up the system. In comparison to a scripted dialogue, where one can immediately start writing the dialogue itself, rule-based dialogue systems require a properly set up knowledge base, sometimes complete ontologies of the world to be able to reason about it. All of these things need to be finished or worked on before the creation process of the dialogue can commence. This leads to the point where rule-based approaches just can not be used in projects with a strict time limit. However, for those projects where time is not of the essence, the coverage one can achieve with rule-based approaches is technically endless.

This ties in to the main limitation we found rule-based approaches to have. They have unlimited potential. Now, you might wonder: *why is that a limitation?* Because of the unlimited potential, it means the scope you set for a project using a rule-based approach is immediately also unlimited. You must define everything, you must think of every case that can happen. To combat this issue we have chosen the social practice theory as our delimiting factor (see chapter 5).

Chapter 5

Social Practice

5.1 Definition

A social practice is not a technique to program dialogue management systems. It is used to give structure to a dialogue in order to make the rule-based approach more applicable. In the background chapter (Chapter 2) a first impression of Social Practice was given. We mentioned we will use the definition as provided by V. Dignum and F. Dignum. [12] and Dignum et al. [11].

V. Dignum and F. Dignum describe the characteristics of social practices as a cooperation of three categories of elements. They make a distinction between the physical elements (Materials), the social elements (Meaning) and the competences. The physical elements consist of the objects in the environment, which they name *resources*, the people and agents in the environment, which they name the *actors*, the *locations* of the elements (physical or temporal), the *activities*, which are potential courses of action. They are tied to the social practice but do not need to be performed. Last in the physical context are the *plan patterns*, which again provide courses of action, but on a larger scale. From these patterns, planning processes can start.

The social elements contain the *social context*, subdivided into the *social interpretation*, which determines the social context in which the practice is used and can vary highly per participant of the social practice. Every participant interprets things in their own way. Furthermore, there are the *roles* and the *norms*. The roles determine somewhat how the participant should act in the social practice or at least what is expected of them. Norms also have a similar purpose, as they give rules of (expected) behaviour in the social practice. Another part of the social elements is *meaning*, which indicates the social effects of the actions that can be performed within the social practice.

The last category, *competences*, describes the skills the participants require to take part in the social practice. For example, a doctor needs to have medical knowledge before they can receive patients in a consultation.

In figure 5.1, we can see the social practice table as seen in the paper by V. Dignum and F. Dignum [12], only altered to contain some example values of a consultation with a patient. There are some interesting parts about this, and some straightforward. An otoscope [25] is one of the tools a doctor can use to examine their patients, as is a reflex hammer and so on. Medical

gloves are also a recognizable resource of a doctor for hygienic purposes. Looking at the table, there seems to be some resemblance between actors and roles, however they are different. Actors have roles, and must have them. However roles must not necessarily have actors. One can bring a relative to the consultation, who will have the role of bystander, however a consultation can also be performed without a bystander. For example, Bob, John and Mary are actors. Bob has the role of doctor. John is a patient and Mary is his wife, thus they have the role of patient and bystander, respectively.

Abstract Social Practice	**Consulting a patient**
Physical Context	
Resources	Otoscope, medical gloves, reflex hammer, ...
Places	Locations of actors and resources
Actors	Doctor, patient, assistant, secretary, ...
Social Context	
Social interpretation	Severity of the complaint, (un)pleasant conversation, ..
Roles	Doctor, patient, bystander, assistant, …
Norms	Being polite, showing empathy, ...
Activities	Sharing information; Analyzing information; Figuring out problem; Analyzing best solution; Sharing solution;
Plan patterns	See figure
Meaning	comfort, understanding, empathy,
Competences	• Medical skills and knowledge • Communication skills • Some understanding of psychology

FIGURE 5.1: The social practice model of a consultation.

Moreover, an actor is unique, but a role can be assigned to two actors: you can bring a relative and a friend to a consultation, both of which would perform the role of bystander, but they would be a unique actor (relative and friend respectively). Social interpretation is an interesting concept. The severity of the complaint is a good example of where interpretation often differs between patient and doctor. The patient often thinks their pain is severe, because they have not seen or experienced more extreme before. The doctor, however, often has seen more extreme cases and is able to rationalize the pain that often causes great emotion in the patient. This, however, is a fine line that doctors must walk and going either way of this line can cause bad results: either the patient dislikes the way you are treating them, or you are overestimating the pain, which might lead to an incorrect diagnosis. This, along with many other reasons, is why learning communication skills is so important for health-care professionals.

In figure 5.2 we see a plan pattern as inspired by the work of Jeuring et al. [20]. It shows the various steps a health-care professional goes through during a consultation (shown as the squares). These steps are parts of a conversation from start to finish. These are connected by arrows which are the ways one can navigate through the conversation steps. For example, going from a greeting and an introduction straight to providing a solution will not work. In this example the 'gathering information' (shown in gray) step has many subsections that are not shown in the diagram. This is to make the diagram more readable, but we will go into them in when discussing the figure below.

We can see the consultation starts off like most normal conversations do between strangers: a greeting followed by an introduction. After which the actual consultation-specific parts begin. The reason for the appointment is determined: what is bothering the patient? After that, the prehistory is determined: When was the last visit? Has the patient ever experienced this before? If so, when was that visit? What treatment was given? All of these questions can help speed up the process, making it an important step in the consultation. If the complaint of the patient seems identical to a previous visit, the doctor will most likely require less information to determine the solution to the problem. The next step is explaining the procedure, which is to manage the expectation of the patient and to put them somewhat at ease. After the procedure is explained and the patient does not have any questions, the main part of the consultation begins: gathering information. Gathering information (shown in gray) consists of many subscenes. However these subscenes are not in any particular order. The doctor can choose to gather information as they wish. The subscenes of gathering information are: influences, experience, connections, functioning, attributes, coping, nature, localization, expectations and time lapse. Each of these scenes are directed at acquiring information to determine the problem of the patient as best as possible.

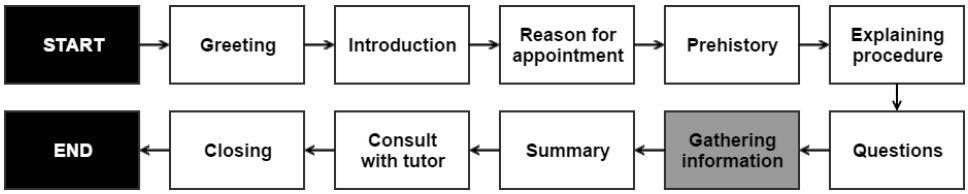

FIGURE 5.2: The plan pattern for a consultation.

Gathering information is required to make a diagnosis after which a solution or treatment can be offered. However, some diagnoses are easier to make than others. So the process of gathering information can be longer or shorter depending on the complaint and the cooperativeness of the patient. This makes the plan pattern of a consultation not as straightforward as the example seen in the work by V. Dignum and F. Dignum [12]. There are most likely even more parts of information that can be gathered than were given, but these are some of the most important ones.

5.2 Expansion

In the previous chapter, we spoke of rule-based dialogue and rule-based approaches. We mentioned as one of the major limitations of these approaches, their unlimited potential. That is where social practices offer a solution. Social practices describe an activity or range of activities that often happen in a society. A social practice could be driving to work, making a phone call, playing football with your friends and so on. All of these activities have some form of social interaction, even if it is not that obvious. Driving to work, although you might drive alone in your car, there will still be other people on the road, with which you exchange signals to show your intention (place on the road, indicating direction, honking your horn). In all of these social practices, there is a finite amount of things to look at. When you are in a consultation with a doctor, you do not have to worry about honking your horn or shooting the football into the right direction. This gives a natural, but necessary boundary to the scope of the project. This in turn means that social practice theory not only solves one of the limitations of rule-based approaches, but also expands on it, by giving more clarity of direction. In essence, the social practice does not provide a hard boundary, but a 'soft' one. It provides the expected behavior, it does not exclude unexpected behavior.

Providing the most likely actions is not the only way the social practice provides a boundary to the potential of rule-based approaches. The interpretation of these actions also becomes more clear if the social practice is clear. For example, raising your hand in a high school class room might mean you have a question, or would like to go to restroom, but raising your hand at an auction would be interpreted as a bid on an item. The exact same action leads to two different interpretations. A person that has the profession of fireman could be in a consultation (maybe they were injured), but they would not perform the role of a fireman. Instead, they are in the role of patient, as they are being treated. Similar to the actions and the interpretation of these actions (raising your hand and its meaning), all the other parts of the social practice model (see figure 5.1) also create a set of possibilities in each of these parts, rather than having to consider each and every possibility in each of these parts: Not all resources should be available or are available in a consultation, not all roles are available (a fireman becomes a patient in this social practice) and so on.

Chapter 6

Implementation

6.1 Structure Overview

Now that we have a clear idea of the theory, it is time to discuss the practical side of things. We have seen chat bots (chapter 2), rule-based approaches (chapter 4) and social practice theory (chapter 5). How do we take these three technologies and put them into one functional design, with which we make the application. A more in-depth discussion on each of the subjects is given in the next few sections. As we discuss these technologies, we will reference the system architecture (Figure 6.1)

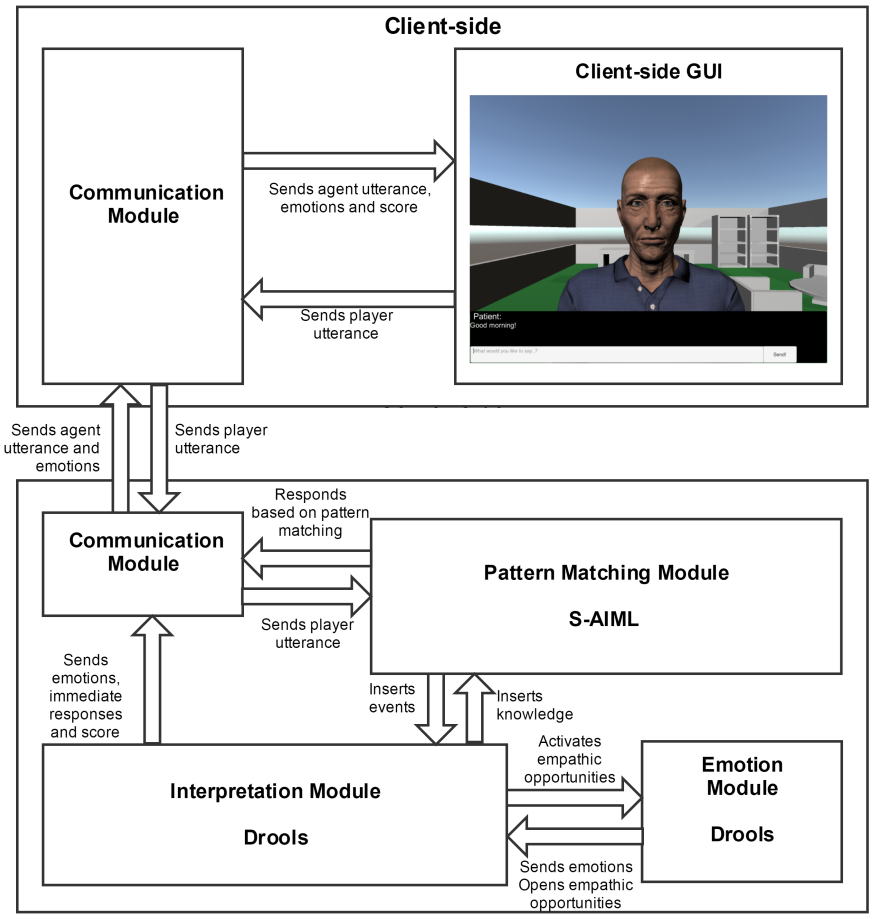

FIGURE 6.1: The system architecture of SALVE.

To start with, we must be able to communicate between a player and our agent. To do that, we must start by processing the input of the player. This is where chat bots come in: AIML allows us to define patterns with which to match the input of the player. The AIML gives a response based on what pattern was matched and what the template response to that pattern is. This gives us input and output.

But giving an automated response to whatever the player says does not seem intelligent at all, nor does it seem human-like. Humans have emotions, share these and almost never give the same response to a question multiple times. In fact, asking the same question to a person over and over, will most likely give you both of these things simultaneously: they will respond with something like "I just told you, ..." or "Did you not hear me, I just said...". These kinds of responses also tend to come paired with emotions like anger or irritation. To implement emotions, we took a look at the OCC model [24], by Ortony, Clore and Collins. We chose to use the original OCC model, however for the interested reader, there were suggestions for a reform by Steunebrink et al. [31]. The emotions as given by the OCC model are implemented by events fired by the AIML, which then trigger a emotion in the emotion module, which in turn is passed back by the interpretation module. Giving a different response to the same question is already in ALICE, the chat bot we use, however we have added extra rules to make sure the right information is still given. More on this in the rules section.

The social practice, as defined in the last chapter, helps us to steer our categories (AIML) and rules into the right direction and limits the amount of work that needs to be done. This social practice was formed by using information from the work by Jeuring et al. [20], combined with the logic as described by V. Dignum and F. Dignum [12].

All of these things happen in the server side of the game. The client simply displays the data given by the server and sends the server the input of the user. The communication between the client and server goes through a web socket.

6.2 Chat bot

We followed the general scheme of AIML, as explained in chapter 2. We will give a short recap, after which we explain how we used this and what additions we made to AIML to create social AIML or S-AIML.

AIML is a XML-dialect, used for building natural language software. It uses pattern matching to match the input, and produce an output based on the matched pattern. This is shown in the structure of the language. The lowest level is a pair of a *pattern* (what the input is compared to) and a *template* (the output, should the pattern match), which combined forms a *category*. A category can be within a topic, or not. A topic allows the chat bot to show more specific behaviour, should the topic be active. This also allows identical patterns to exist. This is often used to give a different response, based on the current topic. For example, given the sentence: 'I

really like comedy', one could think of stand-up comedy, a theatre, a tv-show, or a movie. However, if the topic was set to movies, the chat bot would know that a comedy film was the intended type of comedy, allowing the chat bot to respond with something like "Yeah I really like Liar Liar, have you seen it?". Furthermore, we have the star (*) symbol, which is a wild card symbol that can match to any input, the *that* tag, that allows question-answer modules. The that tag matches the previous sentence of the chat bot. For example, if the chat bot has just asked 'Do you like movies?':

```
<category>
  <pattern> No * </pattern>
  <that> Do you like movies? </that>
  <template> Oh, why not? </template>
</category>

<category>
  <pattern> Yes * </pattern>
  <that> Do you like movies? </that>
  <template> Ah nice, what genre do you like? </
     template>
</category>
```

However, the user could just reply with 'Yeah I do', which would not match with 'Yes *'. This is where the *srai* tag comes in handy. It allows for recursion or redirection of a pattern matching. In this case, we want to match any kind of confirming answer to the template 'Ah nice, what genre do you like?'. So we redirect as such:

```
<category>
  <pattern> Yeah * </pattern>
  <that> Do you like movies? </that>
  <template>
        <srai> Yes * </srai>
  </template>
</category>
```

Doing this with as many other confirming responses, makes the conversation more complete. If you do not catch all of these cases, the user has the possibility of falling through and then any immersion of the user in the conversation will be lost.

To add to the strength of AIML and to improve the communication with the rule-base, we have made some additions to the AIML. We need to be able to communicate from the AIML to the rules and the other way around. This example rule below shows both sides of this communication:

```
<category>
  <pattern>
    My name is *
  </pattern>
  <template>
  <insert packageName="scenarios.large.introduction"
     typeName="NameReceived">
  {
    "name":"<star index="1" />"
  }
  </insert>
    Hello <getDroolsTemplate />, a pleasure to meet
       you. My name is <getDroolsTemplate />.
  </template>
</category>
```

In this category we see two tags that are not yet known. The first tag is *insert* and the second is *getDroolsTemplate*. The insert tag does what it states, it inserts data into the rulebase. In this case it inserts the *NameReceived* event with as its data a name, which is set to the contents of the first * wildcard (there is only one). The second unknown tag is the *getDroolsTemplate* tag. This tag reads from a reaction queue which is filled by the rules. This allows all sorts of processing to be done on the rules side, to then give output based on that processing.

Besides the communication, we need to establish where we are in the conversation. To do this, we established the concept of scenes. A scene consists of a set of categories in AIML and has certain events and rules tied to it in the rules part of the application, which we will discuss in detail in the next section. A scene exists of a certain set of categories, that lead to a local goal. A simple and straightforward example is the greetings scene. It is where greetings are exchanged. Once this is done, the local goal is achieved and the scene is completed. This prompts the next scene to start in a certain order. However, in later scenes there are also categories that are outside the scene, which allow entry into the scene, which in term promotes skipping parts of a conversation or following a different order. This is something people do on a daily basis: You can start a conversation by asking somebody how they are doing, but you can also first say hello and then ask them how they are doing.

6.3 Rules

For the rule-base we chose to use Drools [4, 13]. Drools is a language in which rule-based applications can quickly be made. The way of writing rules is intuitive and is even advocated as something business people can do. Drools simulates a persons mind (in essence) quite well. When your knowledge base contains a fact, perform this action (a condition-action pair). For the purpose of a conversation, this is mostly when your speech partner mentions something, then store the information, respond, add information to the response or show emotion.

In the documentation for Drools, written by Proctor et al. [27], there is a section about the advantages of a rule engine. One of the advantages that stood out for this project is the readability of the rules. Proctor et al. mention it as "*Understandable Rules*". It can be easy for nontechnical domain experts to make them (read health-care professionals and professors).

Another section in documentation even mentions when one should choose a Rule Engine. Two factors stand out here. First, when the logic changes often. This is true in our scenarios, where not only the field is constantly evolving and learning new ways to deal with patients. Not to mention new medicin and treatments that are being made. Besides the change in health-care as a field, there is ofcourse also the change in how new upcoming health-care professionals are taught. Each professor has their own way of teaching their students. Second, one should use a rule engine when domain experts are readily available, but are nontechnical. This is most likely the case as well, assuming the users of this application will be those professors and health-care professionals, who have most likely not seen programcode before.

In the AIML section, we already saw some of the parts of the rule-base and its communication with AIML. The communication towards the rules is done by activating certain events. The moment anything is changed in the knowledge base, the rules are checked and fired if their requirements are active. So if a certain event is activated by the AIML, the rules that require that event to have happened are checked. If there are no other requirements, the rule is fired. Let us look at the example from before, only this time with some of the accompanying rules:

```
<category>
  <pattern>
    My name is *
  </pattern>
  <template>
  <insert packageName="scenarios.large.introduction"
     typeName="NameReceived">
  {
    "name":"<star index="1" />"
  }
  </insert>
  Hello <getDroolsTemplate />, a pleasure to meet you.
      My name is <getDroolsTemplate />.
  </template>
</category>

declare NameReceived
  @role(event)
  name:String
end

rule "NameReceived"
when
  NameReceived($name:name!=null)
  $agent:Person(this isA Agent)
  $player:Person(this isA Player)
then
  $player.setFullname($name);
  reaction.add($name);
  reaction.add($agent.getFullname());
end
```

The AIML activates the NameReceived event via the insert command. It adds the contents of what was in the * wildcard as the 'name' string in the NameReceived event. The rule "NameReceived" checks whether its required facts are satisfied. These facts (in the 'when' section) can be read as follows:

- There is a *NameReceived* fact in the knowledge base with a *name* that is not *null*.
- There is a Person fact in the knowledge base that is a Player.
- There is a Person fact in the knowledge base that is an Agent.

When all of these requirements are fulfilled, then the full name of the player fact is set to the name that was passed via the *NameReceived* event. The name of the player and the full name of the agent are added to the reaction queue. This reaction queue is then read in the template via the command *getDroolsTemplate*. For example, given the name of the bot is John Doe, I could say "*My name is Lucas Weideveld*". Saying that would give me the reply: "*Hello Lucas Weideveld, a pleasure to meet you. My name is John Doe*".

This shows the communication that is possible between the rules and the AIML. However, there are more powerful things we can do using drools. If you are at the doctor and he does not introduce himself. Instead he goes and asks you about your problem, that seems strange. It is custom to know who you are speaking to, before starting a conversation. This is where drools shows one of its strengths:

```
rule "NoName"
salience(2)
when
  not(NameReceived() or NameReceivedAgitated())
  ExitScene(scene.name=="Introduction")
then
  OCCNotHappenedEvent e=new OCCNotHappenedEvent();
  don(e,DesirableEvent.class);
  don(e,ProspectedRelevantEvent.class);
  insert(e);
  controller.respond("Ehm.. sorry, but who are you?");
end

rule "DesirableProspectedEventNotHappened"
when
  $d:OCCNotHappenedEvent(this isA
     ProspectedRelevantEvent, this isA DesirableEvent)
  $agent:Emotion(this isA Agent)
then
  int satisfaction = $agent.getSatisfaction();
  int disappointment = $agent.getDisappointment();
  if (satisfaction > 0) {
    $agent.setSatisfaction(satisfaction - 1);
    controller.setSatisfaction($agent.getSatisfaction
       ());
    controller.print("decreased satisfaction");
  }
  else {
    $agent.setDisappointment(disappointment + 1);
    controller.setDisappointment($agent.
       getDisappointment());
    controller.print("increased disappointment");
  }
end
```

We see here a couple of new aspects of drools. We see the concept of scenes and we see the 'not' keyword. The scenes are, as explained in the previous section, a way to give us a sense of where we are in the conversation. The 'not' keyword can be read as 'there must be none of...'. So in this case there must be no NameReceived or NameReceivedAgitated events before exiting the '*Introduction*' scene. This results in a OCCNotHappenedEvent being made. It is assigned as desirable (you want the doctor to tell you who he is) and prospected relevant (it is expected and relevant to you). This would lead to disappointment according to the OCC model [24] which we can see in the second rule. Either it will decrease satisfaction, which is the opposite emotion, or it will increase dissapointment. Although one might think of a different emotion for this particular case, it is fairly accurate as to what one might feel at such a time. In the last command we see a respond method. This is a way for the rules to directly give a response to the user, bypassing S-AIML as it were. In this specific case, the agent wants to know who he is talking to.

This shows how we have implemented emotions, however we also spoke of the ability to not repeat a recently spoken sentence. Even though this is already present in ALICE on its own, we have also added some rules to add this aspect even more clearly. The concept we added is similar, but not entirely the same. We added an extra comment that when a user goes back to a topic that was already discussed, the patient responds to this, as explained in appendix B.

6.4 Social Practice

As the current version of the implementation is mostly based on verbal communication, the most important part of the social practice is the social context and its meaning. However, the activities and plan patterns give us a good overview of how we can implement our scene structure. The competences of the player is the point that is being tested in this game.

If we look at the example from the last chapter (figure 5.1), we see that the parts of the social context are the interpretation, the roles and the norms. Social interpretation and norms are the more important parts in this. Roles are fulfilled automatically by the game, as these are predefined. Social interpretation however can differ not only per culture, but also per person. This provides a challenge for this game to fully grasp and then overcome. It is impossible for the maker of the rules to be fully objective in this, so there will always be parts of the norms and social interpretation of the creator in the conversation. However, there is a basic set of norms and rules that many people share among a culture, allowing the application to be effective if made by a '*local*'.

The more structural part of the social practice can be more easily translated into the application. Namely, the activities that are in the social practice can be added into the game as rules/AIML. As can be done with the various parts of the plan pattern as we can see from the structure of the code. The

code is structured per scene. The scenes each have their own S-AIML categories (in the Pattern-matching module) and their own set of rules split into the model for the rules and the rules themselves (in the Interpretation module). For example for the current large scenario, inspired by Communicate!, the scenes are as follows:

Introduction, *Prehistory*, *Gathering Information*, *Explaining Procedure*, *Wrapping up Introduction*, *Transition Main Issue*, *Nature*, *Localization*, *Time Lapse*, *Influences*, *Connections*, *Experience*, *Attributes*, *Functioning*, *Coping*, *Expectations* and *Closing*.

Each of these scenes has a level. This level determines whether the player can switch between the different parts of the level without causing confusion. For example, the *Introduction* scene is level 0. The *Functioning* scene is level 2. Going from saying hello to asking how his complaint is affecting his work, is a strange switch in the conversation, which in turn has a negative impact on the final score of the player. For a full overview of the scenes we refer you to the appendix B. The appendix also gives an example of a complete set of rules and S-AIML for the closing scene.

6.5 Scenario walkthrough

In this section we set out a walkthrough of the current doctor-patient scenario we have made. This will contain very specific data and sentences. However, it is easy to add more sentences or change already existing ones. We will pick out a few sentences and their corresponding response here. For the full scenario, we refer you to the appendix C. In the examples below the first utterance (the bullets) are always from the player / doctor. The second utterance (the dashes) are always from the agent / patient.

- Good morning.
  - Hello!

About as simple as it can get. However, just a simple greeting has some extras to it. Because of the various ways to greet another person, these all have to be matched. This is done by using a set. If the player says any of the words in the set, then the pattern will match. For the response we chose to pick randomly from this same set of greetings. So this example of two greetings is but one of many possibilities.

- My name is Chris Vergeer.
  - Hello Chris Vergeer, a pleasure to meet you. My name is Mark van Asperen

This shows an interesting example of the strength of AIML to reuse given data. The name given after 'my name is' is reused in the next sentence. Also the name of the bot is requested from the knowledge base.

- Just need to write down your name and date of birth.
  - So that is Mark van Asperen. I was born on 16-01-1976, doctor.

In this example two pieces of data are retrieved from the knowledge base. The name and the date of birth of the agent.

- I am taking over some patients of doctor Aarts today.
  - How come?
- Do I understand correctly that this annoys you?
  - Well it isn't great. But whatever.. assistants work here more often.

This is one of the examples where the player can anticipate the emotions of the agent. It is quite likely that the patient is not happy that he is not seeing his own doctor. The response can be viewed as being annoyed. This however, shows a strong disadvantage of written word versus spoken. Because there is no tone / emphasis, it is hard to determine if the person is indeed annoyed by the fact that he is seeing an intern.

- Would you like to know or add something?
  - No everything is clear, doctor!

The above sentence pair may seem easy, but it has an entire check list behind it. Because of the walkthrough fully talking about all of the various aspects of the conversation, the patient has no questions. However, should you rush through the conversation, the patient will ask about the aspects that were skipped.

- I will summarize my findings for you.
  - Please do.
- If I understand correctly, you have been having a sore throat for about a week.
  - *nods*
- This is interfering with your night 's rest, leaving you weary.
  - *nods*
- You fear that this might endanger your performance.
  - *nods*
- You would prefer I prevent this by giving you a strong treatment.
  - Exactly, that's it doctor!
- Okay I will have a chat with your GP now.
  - Okay, thank you.

As already discussed in the previous section, more specifically in the appendix B, this example is quite extensive in code. The keywords are matched and determine what the response should be. If all keywords are matched, the final response is given.

6.6 System walkthrough

Now that we have seen how to go through the scenario, let us have a look at how this is processed. Starting from the input coming in to the output coming back. We will look at the server side, as the client simply sends and receives the input and output. Let us take the following example and see how this is processed:

- Would you like to know or add something?
  - No everything is clear, doctor!

To begin, the input is given to the Pattern Matching Module. This checks the AIML for matching patterns. Should a pattern match it will go to the template and reply what is in that. However because of the extensions in S-AIML, the rules might be activated. Let us have a look for this specific example what happens.

```
<category>
  <pattern>
    * like to know *
  </pattern>
  <template>
    <srai>like to know</srai>
  </template>
</category>

<category>
  <pattern>
    like to know
  </pattern>
  <template>
    <srai>anything to add</srai>
  </template>
</category>

<category>
  <pattern>
    anything to add
  </pattern>
  <template>
    <insert packageName="scenarios.large.global"
       typeName="SentenceSpoken" />
    <insert packageName="scenarios.large.
       wrappingupintroduction" typeName="Questions" />
    <getDroolsTemplate />
  </template>
</category>
```

In the code above we can see three categories. The first will match our input, "*Would you like to know or add something?*". This category uses the srai tag to redirect it to the second category. That category also redirects it, to the third category. The third category inserts two events, *SentenceSpoken* and *Questions*, and retrieves a drools template via the *getDroolsTemplate* command. These events are inserted into the Interpretation Module. So let us have a look at how this module handles the events.

```
rule "TooLongSilence"
when
  $s:SentenceSpoken()
  not(SentenceSpoken(this != $s, this after[0ms,40s] $
     s))
then
  int r = new Random().nextInt(5);
  String response = "";
  switch (r)
  {
    case 0:
    {
      response = "Are you going to say something?";
      break;
    }
    case 1:
    {
      response = "Hello..?";
      break;
    }
    case 2:
    {
      response = "Could we move this along? I don't
         have all day.";
      break;
    }
    case 3:
    {
      response = "Are you thinking or..?";
      break;
    }
    case 4:
    {
      response = "Are you still there?";
      break;
    }
  }
  controller.respond(response);
end
```

The above code sample shows the rule that activates on the insertion of the *SentenceSpoken* event. Reading through the 'when' section of the rule, it might seem confusing. However the activation is actually quite logical. The rule activates when a SentenceSpoken event is active (these are produced by every sentence from the player) and there is no other SentenceSpoken event for fourty seconds after that. The 'then' section of the rule simply selects one out of the five responses and responds that to the player. Of course fourty seconds is an arbitrary amount of time and this can be changed. The goal of this rule is to add a bit of 'humanlike' behavior to the agent. A normal human would also get impatient if you suddenly stopped talking to them.

```
rule "NoQuestions"
salience(7)
when
  Questions()
  DoctorExplanation()
  RecognizedPatient()
  LastVisit()
  DataRequested()
  UrgentMeeting()
  ShareFindings()
  Procedure()
then
  Score scoreReasoningTemp = (Score)drools.
     getWorkingMemory().getGlobal("scoreReasoning");
  scoreReasoningTemp.add(new Pair(2, "Wrapping up
     introduction - You were able to answer all the
     patients questions or able to have everything
     clear before they asked a question."));
  drools.getWorkingMemory().setGlobal("scoreReasoning"
     , scoreReasoningTemp);

  reaction.add("No everything is clear, doctor!");
  OCCHappenedEvent e=new OCCHappenedEvent();
  FinishedSceneEvent f = new FinishedSceneEvent();
  f.setSceneName("WrappingUpIntroduction");
  insert(f);
end
```

The above code sample shows the rule that is activated because of the insertion of the *Questions* event. Because of the way the walkthrough is done, all of the other events are also in the knowledge base, making this the activated rule. Interestingly, if one of the other events is missing, other rules are chosen (see below). However, the above rule is activated in the walkthrough, so let us have a look at what happens when it is activated, by looking at the 'then' section. First a score pair is added to the score list. The player receives two points (again arbitrary number) and a explanation is stored as to why this score was given.

Next, the reaction is added *'No everything is clear, doctor!'*, which is the exact reaction seen in the response on the client side. The final act of this rule is to add a *FinishedSceneEvent* to the knowledgebase for the scene *WrappingUpIntroduction*. We will explain what this effects a bit later on.

```
rule "DoctorExplanationQuestion"
salience(6)
when
  $q:Questions()
  not(DoctorExplanation())
then
  controller.print("added DoctorExplanationQuestion");
  reaction.add("Yeah why am I not talking to my own GP
     ?");
  retract($q);
end

rule "RecognizedQuestion"
salience(5)
when
  $q:Questions()
  DoctorExplanation()
  not(RecognizedPatient())
then
  controller.print("added RecognizedQuestion");
  reaction.add("Yeah you are aware I am a patient of
     doctor Aarts, right?");
  retract($q);
end

rule "LastVisitQuestion"
salience(4)
when
  $q:Questions()
  DoctorExplanation()
  RecognizedPatient()
  not(LastVisit())
then
  controller.print("added LastVisitQuestion");
  reaction.add("Yeah, do you know about my last visit?
      It was for a persistent throat infection.");
  retract($q);
end
```

```
rule "DataRequestedQuestion"
salience(3)
when
  $q:Questions()
  DoctorExplanation()
  RecognizedPatient()
  LastVisit()
  not(DataRequested())
then
  controller.print("added DataRequestedQuestion");
  reaction.add("Not really.. But don't you normally
     ask for my date of birth and such?");
  retract($q);
end

rule "ShareFindingsQuestion"
salience(2)
when
  $q:Questions()
  DoctorExplanation()
  RecognizedPatient()
  LastVisit()
  DataRequested()
  not(ShareFindings())
then
  controller.print("added ShareFindingsQuestion");
  reaction.add("Yeah, is doctor Aarts going to be
     aware of what we discussed? I don't want to have
     to explain this to him next time I see him");
  retract($q);
end

rule "ProcedureQuestion"
salience(1)
when
  $q:Questions()
  DoctorExplanation()
  RecognizedPatient()
  LastVisit()
  DataRequested()
  ShareFindings()
  not(Procedure())
then
  controller.print("added ProcedureQuestion");
  reaction.add("Yeah what is the procedure going to be
      here?");
  retract($q);
end
```

Based on which event is missing one of the above rules is chosen. All of the these rules will add a reaction to the queue (a question) and then retract (remove) the *Questions* event from the knowledge base. This allows the doctor to keep asking whether the patient has questions until he does not have any questions anymore.

```
rule "FinishedScene"
when
  FinishedSceneEvent($scene:sceneName)
  $sp:CouncilWithGP()
then
  int level = -1;
  if (l0.contains($scene))
    level = 0;
  if (l1.contains($scene))
    level = 1;
  if (l2.contains($scene))
    level = 2;
  if (l3.contains($scene))
    level = 3;

  String sceneName = getSceneToGoTo(level, $scene, l0,
      l1, l2, l3, l0F, l1F, l2F, l3F);
  if (sceneName.equals("FINISHED"))
  {
    Score scoreReasoningTemp = (Score)drools.
       getWorkingMemory().getGlobal("scoreReasoning");
    controller.respond("123GOTOSCORESCREEN123");
    controller.print("" + scoreReasoningTemp.size());
    for (Pair pair : scoreReasoningTemp.getScore())
    {
      controller.print("responded an entry");
      controller.respond("123SCORE123:" + pair.
         getScore() + ":" + pair.getReason());
    }
    return;
  }
  ChangeOfScene change = new ChangeOfScene();
  Scene scene = (Scene)$sp.getScenes().get(sceneName);
  change.setToScene(scene);
  insert(change);
end
```

A bit earlier in this section we saw the rule *'NoQuestions'* which inserted a *FinishedSceneEvent*. Let us have a look at what the effects are from this event. We will not go all the way into the system, but we will explain the general way this is handled.

In this rule we see the levels show up (in the 'when' section). In this rule we see the level of the given scene is determined first. Once this is done the the function 'getSceneToGoTo' is called. This function simply determines the next scene to go, by following this scheme:

- Are all scenes in the given level complete?
- Yes - Go to the first scene in the next level.
- No - Go the first scene that is not completed in the current level.

This scheme has one sidemark. If all the scenes in the final level are completed, the scene name 'FINISHED' is returned. As we can see in the rule above, this results in the system sending a message relaying the scores of the player to allow for viewing in the client.

If the game is not over (the scene name is not 'FINISHED'), the final part of the rule is completed, which sets in motion the change of scene to the scene given by the algorithm.

Chapter 7

Comparison

7.1 Reasoning

Comparing two applications that seemingly have no similarity besides their subjects might seem strange. However, even though the rest of the paper might lead you to think that our application is '*better*' than Communicate!, we are convinced that both applications have their strengths and weaknesses. We believe that they both can serve their purpose and teach people meaningful concepts with which they can improve upon their communication skills. We will provide a slight recap and then list some advantages and limitations of both approaches.

7.2 Communicate!

Communicate! [20] is a serious game made to improve the communication skills of health-care professionals. It comes in the form of a scripted dialogue where the user plays the role of the doctor. The game performs the role of the patient. In each point of the dialogue, the player must choose between a number of responses, with some freedom of interleaving between the various subjects. The game is played in a browser and allows creation of dialogues via the editor provided with the game. There are various extra ways of personalizing the dialogue and the outcome. One can choose the patient avatar and provide their own parameters and thus scoring parameters.

7.2.1 Advantages

The advantages of Communicate! are easily found. With its strong editor and freedom of parameter choice, the customizability of the game is very strong. This allows health-care professors to determine nearly everything about the dialogue they want to teach their students. Furthermore, because of the editor being so well-developed, the speed at which such a dialogue can be created (or altered) is also impressive. This allows a teacher to stay in line with any changes in the material of the course. The strong editor is not the only reason for the speed at which these dialogues can be developed. The choice for scripted dialogue also adds to this speed, as scripted dialogues are generally made quicker than rule-based or other approaches as long as the scenarios remain small.

Besides the choices that increase the speed, the game was also made to work in the browser, which allows for a good coverage of availability. Nearly any computer can run the game in the browser, rather than requiring to install something. Because of the parameter system, the scoring with which the teacher can rank their students is also well-established. The teacher can not only decide the parameters but also see the results in the game itself in the score screen.

7.2.2 Limitations

The game does have its limitations. The effort put into creating a scenario in scripted dialogues is basically 1 to 1. When adding a sentence, you add a sentence. This makes large scenario's require a great amount of effort to create, in comparison to other approaches. Another limitation is the way of communicating. It is forced by the creator of the scenario, rather than by the player. This not only limits the freedom of the player, but also the criteria of communication skills [16] they can learn from the program as discussed in chapter 3. Also mentioned in chapter 3 was the creation of dialogue which is in important aspect of dialogue as emphasized by Clarke as well [10]. One final limitation is that in Communicate! the creator has to control the score given per dialogue move. Once multiple criteria are used to control the score, it is hard to keep the scoring logical. This takes extra effort to make sure that all moves are scored 'fairly'.

7.3 Our Approach

Our approach uses three concepts to come to a serious game with the aim of increasing the communication skills of the player. The three concepts have been explained: chat bots (ALICE / AIML) in chapter 2, rule-based dialogue in chapter 4 and social practice theory in chapter 5. These combined make a serious game where the player walks through a conversation, giving their own input with the keyboard. The player then receives a response to their input from the agent that is either automated by pattern-matching in AIML or generated by some rule or parameters to be specific to the situation. Agents show emotion, respond to empathic responses and to repetition.

7.3.1 Advantages

The main advantage of our approach is the coverage that is possible with the approach. Because of the boundary the social practice supplies, the amount of work is limited, yet covering of that specific social practice. The possible coverage is endless. The time required to cover a specific social practice is not 1 to 1, as it is in scripted approaches. Because of the strength of the rule-based approach combined with AIMLs *srai* tag, the ability to create references is massive, which allows the ratio of effort to coverage to be better than 1 to 1. This can be deduced by simple logic (see appendix A), however calculating the actual ratio is somewhat troubling. In appendix A, we can also see how AIML provides the ability to avoid problems that might occur when changing a certain piece of content or adding content. In short, AIML provides modularity. The rules on the other hand provide the

ability to step in when the user does not abide by the standards of a conversation. Jumping from subject to subject or simply not saying anything at all. A chat bot would wait until they received a message. A human would get bored or agitated and most likely ask them to hurry up.

7.3.2 Limitations

One of the main limitations of our approach is the availability of the game. It is made in unity, requiring it to be installed and played locally. It is also set up with a client-server model, requiring it to have good server support. Currently it is lacking a good editor, making it so the user is required to have knowledge of the programming languages used to be able to alter the scenarios. The general conclusion that can be drawn from these limitations is that the game and its supporting applications are either not finished or not polished fully. This is however to be expected in such an early iteration of the project. Another downside of the game, which is a part of the future work section, is the lack of speech. Because of written communication a lot of information is lost. Communication through speech recovers a lot of this information, but it also comes with a lot of extra work, which is too much for the scope of this project.

7.4 Practical examples

There are a few practical examples that will help make clear some of the advantages of SALVE over Communicate!. In the setting of a conversation between a doctor and patient, the terminology one uses is quite important. For example, when a doctor speaks to another doctor, he would use medical terms (Latin terms or specific names of medication). For example '*nephritis*' (Inflammation of the kidneys). A patient would not know what that means. A doctor probably would know. If a doctor were to have to explain to a patient that they have nephritis. In Communicate! the player (doctor) would get a limited set of ways to say this, where they would most likely choose the one that seems the most likely to be correct (the one that does not contain the medical term). In SALVE however, they would have to decide for themselves how they would phrase the sentence and also which term to use. This leads to a better learning experience, because they have to make the choice themselves. This is underlined by the work by Cheng [9], where a set of students whose first language was Mandarin Chinese took an English test. They scored highest on multiple-choice cloze (MCC), followed by traditional multiple-choice and lastly the open-ended questions (OE). Cheng mentions: "A posttest survey revealed that 97% of the test takers preferred the MC format because the given alternative answers facilitated comprehension of spoken stimuli and greater accuracy in guessing."

Another practical example actually comes from the current scenarios implemented in SALVE. In the first scenario, the player (doctor) has a conversation with the agent (patient). In this, the doctor questions the patient to receive the information required to make a good diagnosis. He then tests whether he has the required information by running this by the patient. This is in the closing scene (see appendix B). In this process, the doctor can

forget one or two mentioned complaints and then the patient can remind him of this, because the patient knows what is bothering him. Then in the second scenario, the doctor has to report to another doctor, his supervisor, as he is in training. The supervisor does not know what is troubling the patient, so if the doctor misses any part of information, the supervisor might give a incorrect recommendation because of this missing information.

In SALVE the above example would require the player to remember the things that are troubling the patient and make sure that they understand everything correctly. Then they will most likely rephrase that when they speak to the doctor. In Communicate!, however, the player would most likely just have to remember the amount of items there are and then pick between "That was all, what do you think I should give them?" or "There's more.." followed by a choice between some predefined set of conditions. Or possibly even simpler, the player can just pick anything until there is only one option left which is to ask for the supervisor's opinion. Figure 7.1 shows one of the possible ways of implementing this in Communicate! In the end, no matter how complicated the scenario is made in Communicate!, there is some sort of scripting which takes away parts of the tasks that a doctor should be able to do and therefore should learn.

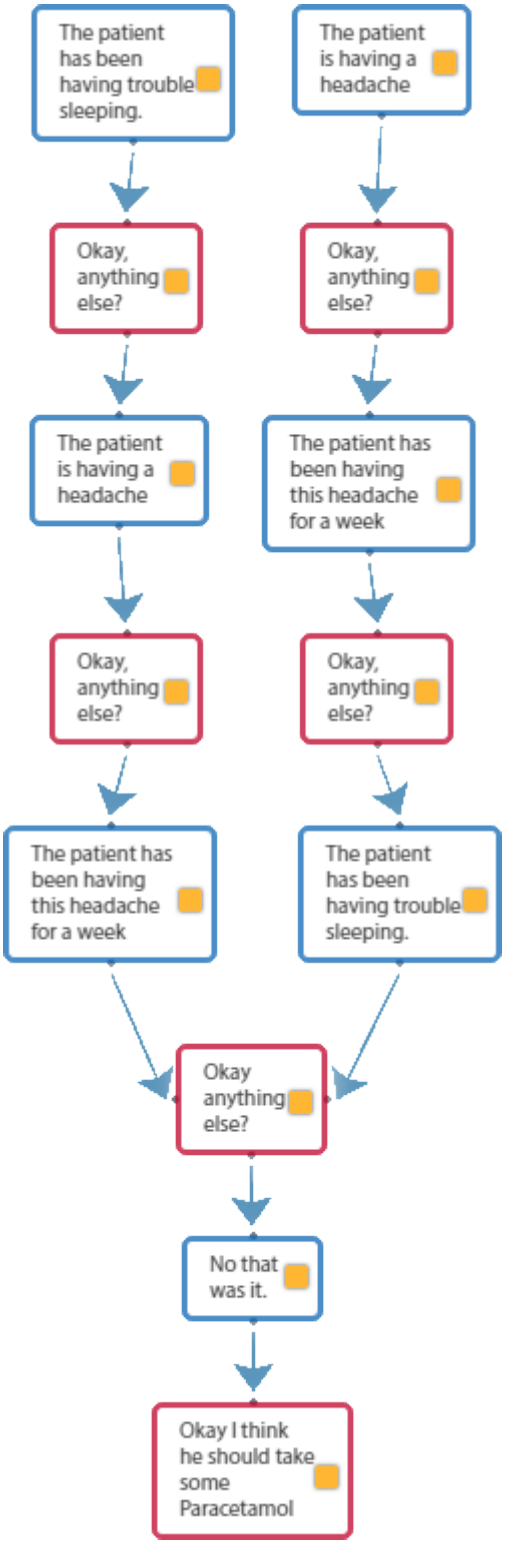

FIGURE 7.1: A possible supervisor consultation in Communicate!

Chapter 8

Conclusion

8.1 Conclusion

We have discussed the theory behind a new application for learning communicative skills. The application focuses on health-care professionals, like its inspiration, Communicate!. The game can of course be made suitable for other scenarios. We have provided a short description of how these theories translate into code. However, as providing a full code-base is not possible in such a thesis, and would likely start bothering readers, we have chosen to provide the fundamental concepts that make it possible to create an agent capable of being a admirable sparring partner for learning communicative skills. One of the main concepts that was discussed was the criteria introduced in chapter 1, as they are considered to be required to become competent at communicating [16]. These were discussed throughout the work and it was concluded that Communicate was lacking in these criteria, compared to a more unrestricted approach like ours. Because Communicate! is scripted, it is impossible to teach some of the criteria. In our approach, the freedom of speech allows the player to learn these criteria.

We provided a comparison between our new approach and Communicate! We provided some practical examples in which SALVE outshines Communicate! The primary conclusion we draw from the comparison, is that the capabilities of our approach exceed those of Communicate on larger projects. We believe that in smaller projects, Communicate! will be able to provide a scenario quicker, that might still fulfill the requirements of the user. However, when looking at large-scale scenario's the modularity of our approach makes it become more powerful than Communicate! We also believe that because Communicate! is further along its development, it is far more polished and user-friendly. Despite that, we still believe that once our approach reaches maturity, it will outgrow the potential of Communicate! and become a very good addition to the health-care field.

We think this study will be significant to the people in the artificial intelligence field, as it is directly related to artificial intelligence. It will also give the first practical view of how different social practices, and thus social contexts, can change an agent's attitude and responses. This study can be of use to the robot industry and online chatbots due to the innovative way of representing data. In a more distant relationship, this study could help the psychology field, by giving a concrete example of how people respond to an agent using this new technique. The most obvious and direct link is that

to the health-care education sector. Both the teachers and students of this sector can benefit from this work.

8.2 Future Work

There are many possible changes that will improve on the work as presented in this paper. Communication is something we all partake in every day and comes in many different forms. Focusing on the most important one for health-care professionals, the face-to-face conversation, we can analyze the improvements that need to be made to our approach for it to approach the complexity of a 'simple' face-to-face conversation. One of the main differences is speaking versus typing. Speaking and understanding speech is a very important aspect of a conversation. It introduces a large amount of extra aspects of linguistics: tone, intonation, volume, word stress, dialect, interpretation, pronunciation, mishearing something and so on. Even though all of these aspects introduce a lot of extra difficulty, they are also key aspects of learning to communicate and therefore should be covered. Another important aspect of the face-to-face conversation is body language. Although it is hard to implement this well in a game, it is a possibility for future work.

As mentioned in the conclusion, the user-friendliness of the game should be increased. One of the main aspects where we can learn from Communicate! The editor and the ability to play online is a good way to improve on our approach in the future.

Besides this, the structure of the application can be improved further by iterating on the strengths of the system. S-AIML can be improved to better capture the essence of what the user is trying to say. Both in oral communication and written communication, there are errors (intentional or not). These should be filtered out as much as possible. The rule-base can be improved upon by more easily allowing a nontechnical person to add to it or create a new rule-base.

Besides some of the more practical improvements that can be made, we can also improve upon the knowledge and the sentience of the agent by adding motives, specifying its identity, values, norms, habits and finally its social practices. These concepts come from the work by Dignum et al. [11], which was briefly explained in section 2.3. This also allows users to more easily define the agent. For example: rather than having to implicitly set the goal of the agent through rules, they can then explicitly set the goal. This goal would be something for the achievement motive. Its affiliation motive could be set to having a good relationship with the doctor. The avoidance motive could be set to avoid situation where the agent feels uncomfortable.

Appendix A

Advantages of AIML

A simple *'proof'* of why AIML alone can make the ratio of creating rules to their coverage better than 1 to 1. Say you want to make a small encyclopedia chatbot. In scripted dialogue, one would have to write each and every possible question one could ask:

- What is ...?
- Tell me what ... is.
- Could you tell me what ... is?
- Tell me about ...

And this list could go on forever if you are creative. Not to mention that for each of these types of questions, we also have to add a subject about which they ask the question. Now in AIML, we can quite easily make use of the SRAI tag to handle this. We only have to write the types of questions once and the types of subjects once. Let us take the example above and add the subjects *a cat* and *a dog*. Here we already see that asking about a plural subject requires the question to be changed: 'what *are* dogs?', 'What *are* cats?' instead of *'what is ...'*. So let's see how the example works out:

```
<category>
  <pattern> What is * </pattern>
  <template> That is <sr/> </template>
</category>

<category>
  <pattern> Tell me what * is</pattern>
  <template>Okay, that is <sr/> </template>
</category>

<category>
  <pattern> Could you tell me what * is</pattern>
  <template>Sure, that is <sr/> </template>
</category>

<category>
  <pattern> Tell me about * </pattern>
  <template> Alright, that is <sr/> </template>
</category>
```

```
<category>
  <pattern> a cat </pattern>
  <template>
  a small domesticated carnivorous mammal
  (Felis catus), kept as a pet and as catcher of
  vermin, and existing in a variety of breeds.
  </template>
</category>

<category>
  <pattern> a dog </pattern>
  <template>
  a domesticated carnivorous mammal (Canis familiaris
  syn. Canis lupus subsp. familiaris) occurring as
  a wide variety of breeds, many of which are
  traditionally used for hunting, herding, drawing
  sleds, and other tasks, and are kept as pets.
  </template>
</category>
```

Now we see what was said before that there is only one category for each question and one for each subject. Now say somehow the amount of effort for scripted dialogues was equal to the amount of effort required to make this example in AIML. Then we will add one more subject: *'a fish'*. First the impact in AIML:

```
<category>
  <pattern> a fish </pattern>
  <template>
  any of numerous cold-blooded aquatic vertebrates
  characteristically having fins, gills, and a
  streamlined body and including the bony fishes,
  such as catfishes and tunas, and the
  cartilaginous fishes, such as sharks and rays.
  </template>
</category>
```

We see that adding only this category makes the example work. Now in scripted dialogues we would have to fully write out each question:

- Q: What is a fish?
- A: That is any of numerous cold-blooded aquatic vertebrates characteristically having fins, gills, and a streamlined body and including the bony fishes, such as catfishes and tunas, and the cartilaginous fishes, such as sharks and rays.
- Q: Tell me what a fish is.

- A: Okay, that is any of numerous cold-blooded aquatic vertebrates characteristically having fins, gills, and a streamlined body and including the bony fishes, such as catfishes and tunas, and the cartilaginous fishes, such as sharks and rays.
- Q: Could you tell me what a fish is?
- A: Sure, that is any of numerous cold-blooded aquatic vertebrates characteristically having fins, gills, and a streamlined body and including the bony fishes, such as catfishes and tunas, and the cartilaginous fishes, such as sharks and rays.
- Q: Tell me about a fish.
- A: Alright, that is any of numerous cold-blooded aquatic vertebrates characteristically having fins, gills, and a streamlined body and including the bony fishes, such as catfishes and tunas, and the cartilaginous fishes, such as sharks and rays.

The amount of typing required to make all of these questions far exceeds that of the AIML example. In most cases, copying and pasting the response is faster, which as a programmer should always make you worried: what happens if the definition of a fish changes slightly. In AIML you would have to alter it in one position. In scripted dialogues, you would have to change it in N positions, where N is the amount of question types. A slight overview of the amount of work for both approaches where N is the amount of question types and M is the amount of subjects in the encyclopedia can be seen in table A.1.

Task	**Scripted Dialogue**	**AIML**
Changing a question	Change M questions	Change 1 pattern
Adding a question	Add M questions and answers	Add 1 category
Changing an answer	Change N answers	Change 1 template
Adding a subject	Add N questions and answers	Add 1 category

TABLE A.1: Changes required for each task

Appendix B

Overview of Scenes

In code snippet B.1 we can see the S-AIML of the closing scene. Here we can see a few interesting aspects. The scene is enclosed in a <*scene*> tag. However, this scene tag does not enclose all the categories. Actually there are double categories inside and outside of the scene. The categories outside of the scene all point to one category (via <srai>). That category then has a insert ForcedChangeOfScene insertion with as a parameter this scene's name (Closing). This allows this scene to be entered from other scenes. This is done via the *forcedChangeScene* rule, as can be seen in code snippet B.2. This rule determines what the consequence is of the scene switch and applies it. The consequences are determined in the function *ConsequencesOfSceneSwap*. The simple description of this function is: if the level of the two scenes is different, the effect is negative, otherwise it is zero. If the given consequence of the function is "s-1" (shortcut for satisfaction - 1), this is executed. Besides the negative impact on satisfaction, the patient also gives an immediate response *"That was a sudden switch in the conversation though."*. This will give the player an idea that this was unwanted behaviour. Here we also see some extra behaviour. If the scene was already completed, an extra response is given: *"Wait, didn't we already discuss this..?"*. Because of the ability to swap between scenes, one could finish a topic and then return to it. Which is generally uncommon and unwanted behaviour.

CODE B.1: S-AIML of the closing scene.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml>
  <!-- CLOSING -->
  <scene name="Closing">
    <category>
      <pattern>
        Summarize my findings
      </pattern>
      <template>
        Please do
        <insert packageName="scenarios.large.global"
           typeName="SentenceSpoken" />
      </template>
    </category>

    <category>
      <pattern>
        Summarize my findings *
```

```
    </pattern>
    <template>
      <srai>Summarize my findings</srai>
    </template>
  </category>

  <category>
    <pattern>
      * Summarize my findings
    </pattern>
    <template>
      <srai>Summarize my findings</srai>
    </template>
  </category>

  <category>
    <pattern>
      * Summarize my findings *
    </pattern>
    <template>
      <srai>Summarize my findings</srai>
    </template>
  </category>

  <category>
    <pattern>
      Summarize the findings
    </pattern>
    <template>
      <srai>Summarize my findings</srai>
    </template>
  </category>

  <category>
    <pattern>
      Summarize the findings *
    </pattern>
    <template>
      <srai>Summarize the findings</srai>
    </template>
  </category>

  <category>
    <pattern>
      * Summarize the findings
    </pattern>
    <template>
      <srai>Summarize the findings</srai>
    </template>
  </category>
```

```
<category>
  <pattern>
    * Summarize the findings *
  </pattern>
  <template>
    <srai>Summarize the findings</srai>
  </template>
</category>

<category>
  <pattern>
    Summary
  </pattern>
  <template>
    <srai>Summarize my findings</srai>
  </template>
</category>

<category>
  <pattern>
    Summary *
  </pattern>
  <template>
    <srai>Summary</srai>
  </template>
</category>

<category>
  <pattern>
    * Summary
  </pattern>
  <template>
    <srai>Summary</srai>
  </template>
</category>

<category>
  <pattern>
    * Summary *
  </pattern>
  <template>
    <srai>Summary</srai>
  </template>
</category>

<category>
  <that>Please do</that>
  <pattern>
    *
  </pattern>
  <template>
```

```
      <insert packageName="scenarios.large.closing"
         typeName="Summary" >
      {
          "summary": "<star index="1" />"
      }
      </insert>
      <insert packageName="scenarios.large.global"
         typeName="SentenceSpoken" />
      <getDroolsTemplate />
    </template>
  </category>

  <category>
    <that>*nods*</that>
    <pattern>
      *
    </pattern>
    <template>
      <insert packageName="scenarios.large.closing"
         typeName="Summary" >
      {
        "summary": "<star index="1" />"
      }
      </insert>
      <insert packageName="scenarios.large.global"
         typeName="SentenceSpoken" />
      <getDroolsTemplate />
    </template>
  </category>

  <category>
    <that>I believe you already said that</that>
    <pattern>
      *
    </pattern>
    <template>
      <insert packageName="scenarios.large.closing"
         typeName="Summary" >
      {
        "summary": "<star index="1" />"
      }
      </insert>
      <insert packageName="scenarios.large.global"
         typeName="SentenceSpoken" />
      <getDroolsTemplate />
    </template>
  </category>

  <category>
    <that>Exactly, that's it doctor</that>
    <pattern>
```

```
      supervisor
    </pattern>
    <template>
      Okay, thank you.
      <insert packageName="scenarios.large.closing"
         typeName="Finished" />
    </template>
  </category>

  <category>
    <pattern>
      supervisor *
    </pattern>
    <template>
      <srai>supervisor</srai>
    </template>
  </category>

  <category>
    <pattern>
      * supervisor
    </pattern>
    <template>
      <srai>supervisor</srai>
    </template>
  </category>

  <category>
    <pattern>
      * supervisor *
    </pattern>
    <template>
      <srai>supervisor</srai>
    </template>
  </category>

</scene>

<category>
  <pattern>
    Summarize my findings
  </pattern>
  <template>
    Please do
    <insert packageName="scenarios.large.global"
       typeName="SentenceSpoken" />
    <insert packageName="scenarios.large.global"
       typeName="ForceEnterScene" >
    {
      "sceneName": "Closing"
    }
```

```
    </insert>
  </template>
</category>

<category>
  <pattern>
    Summarize my findings *
  </pattern>
  <template>
    <srai>Summarize my findings</srai>
  </template>
</category>

<category>
  <pattern>
    * Summarize my findings
  </pattern>
  <template>
    <srai>Summarize my findings</srai>
  </template>
</category>

<category>
  <pattern>
    * Summarize my findings *
  </pattern>
  <template>
    <srai>Summarize my findings</srai>
  </template>
</category>

<category>
  <pattern>
    Summarize the findings
  </pattern>
  <template>
    <srai>Summarize my findings</srai>
  </template>
</category>

<category>
  <pattern>
    Summarize the findings *
  </pattern>
  <template>
    <srai>Summarize the findings</srai>
  </template>
</category>

<category>
  <pattern>
```

```
    * Summarize the findings
  </pattern>
  <template>
    <srai>Summarize the findings</srai>
  </template>
</category>

<category>
  <pattern>
    * Summarize the findings *
  </pattern>
  <template>
    <srai>Summarize the findings</srai>
  </template>
</category>

<category>
  <pattern>
    Summary
  </pattern>
  <template>
    <srai>Summarize my findings</srai>
  </template>
</category>

<category>
  <pattern>
    Summary *
  </pattern>
  <template>
    <srai>Summary</srai>
  </template>
</category>

<category>
  <pattern>
    * Summary
  </pattern>
  <template>
    <srai>Summary</srai>
  </template>
</category>

<category>
  <pattern>
    * Summary *
  </pattern>
  <template>
    <srai>Summary</srai>
  </template>
</category>
```

```
</aiml>
```

CODE B.2: Rule to determine consequences of scene swapping.

```
rule "forcedChangeScene"
when
  ForcedChangeOfScene($next:toScene)
  Agent($currentScene:currentScene)
then
  ExitScene es=new ExitScene();
  es.setScene($currentScene);
  insert(es);

  EnterScene e=new EnterScene();
  e.setScene($next);
  insert(e);
  String consequences = ConsequencesOfSceneSwap($
     currentScene.getName(), $next.getName(), l0, l1,
     l2, l3);
  controller.print("current scene: " + $currentScene.
     getName() + " - going to scene: " + $next.getName
     () + " - CONSEQUENCES: " + consequences);
  if (consequences.equals("s-1")) {
    OCCHappenedEvent event = new OCCHappenedEvent();
    don(event, UndesirableEvent.class);
    don(event, ProspectedIrrelevantEvent.class);
    insert(event);
    controller.respond("That was a sudden switch in
       the conversation though.");
  }
  if (alreadyCompleted($next.getName(), l0, l1, l2, l3
     , l0F, l1F, l2F, l3F)) {
    controller.respond("Wait, didn't we already
       discuss this..?");
  }
end

function String ConsequencesOfSceneSwap(String from,
   String to, List l0, List l1, List l2, List l3) {
  if (l0.contains(from) && l0.contains(to))
    return "0";
  if (l1.contains(from) && l1.contains(to))
    return "0";
  if (l2.contains(from) && l2.contains(to))
    return "0";
  if (l3.contains(from) && l3.contains(to))
    return "0";
  return "s-1";
```

}

In the current architecture of SALVE, we used levels to determine the flow of the conversation (first mentioned in section 6.4). These levels can be seen in figure B.1. The user is allowed to switch between them, but as explained before, the forcedChangeScene rule (Code B.2) will give a response when this is done between different levels.

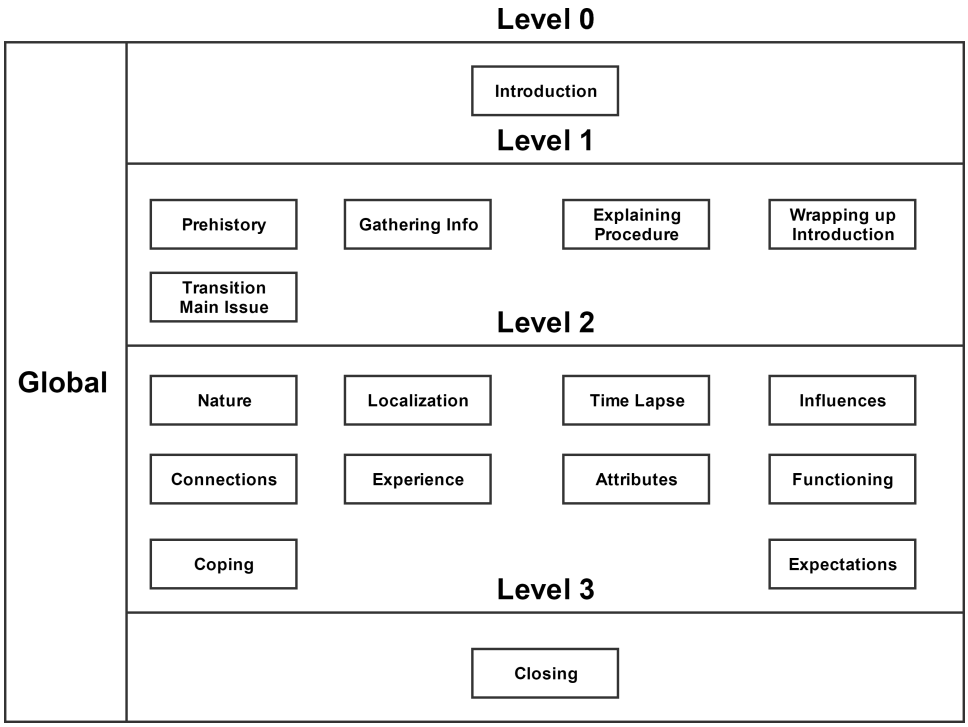

FIGURE B.1: Overview of the levels and the scenes in them.

Now that we have seen the scene swapping and the S-AIML of the Closing scene, let us also have a look at the rules of the Closing scene (code snippet B.3). We see one small rule, which initiates a global list that is used to track the matched items. Then the main rule that handles the summary of the player. There we see a set of keyword lists that are initialized for each of the types of keywords: throat, weary (symptoms), duration, treatment (what does the patient expect) and performance (what is troubling the patient). Each of these lists has a certain amount of different ways to say the same thing. Then all of the keyword lists are looped through to determine if there is a match. Also a check is done to see if this match has occurred before. If so, the repeat flag is turned to true. If a match is found, the matched flag is turned to true. In the final if/else statements, the response is given based on which flags are marked and how many matches have been found so far. If the player says something completely incorrect, the summary loop is broken by a request to speak with his own GP. The final rule at the bottom is a simple one. Once the finished event (see the S-AIML category with the template 'supervisor' B.1) is entered, a FinishedSceneEvent is inserted. This in turn will activate a different rule that will handle going to the next scene. In this specific case, because it is the closing scene that is finished, it will go to a score screen.

CODE B.3: Rules of the Closing scene.

```
rule "InitMatches"
when
  EnterScene(scene.name=="Closing")
then
  matches = new ArrayList<Integer>();
  drools.getWorkingMemory().setGlobal("matches",
     matches);
  controller.print("initialized matches.");
end

rule "CheckSummary"
when
  Summary($summary:summary!=null)
then
  List<List<String>> keywords = new ArrayList<List<
     String>>();
  List<String> throat = new ArrayList<String>();
  List<String> duration = new ArrayList<String>();
  List<String> weary = new ArrayList<String>();

  List<String> performance = new ArrayList<String>();
  List<String> treatment = new ArrayList<String>();

  throat.add("sore throat");
  throat.add("pain in throat");

  duration.add("week");
  duration.add("seven days");
  duration.add("7 days");

  weary.add("weary");
  weary.add("tired");
  weary.add("exhausted");
  weary.add("fatigued");
  weary.add("sleepy");

  performance.add("endanger your performance");
  performance.add("endanger the performance");
  performance.add("performance at risk");
  performance.add("gig at risk");
  performance.add("endanger your gig");
  performance.add("endanger the gig");

  treatment.add("strong treatment");

  keywords.add(throat);
  keywords.add(duration);
  keywords.add(weary);
```

```
keywords.add(performance);
keywords.add(treatment);
Boolean matched = false;
Boolean repeat = false;

for(int i = 0; i < keywords.size(); i++)
{
  List<Integer> tempMatches = (List<Integer>)drools.
     getWorkingMemory().getGlobal("matches");
  if (tempMatches.contains(i))
  {
    repeat = true;
    continue;
  }
  List<String> list = keywords.get(i);
  for(int j = 0; j < list.size(); j++)
  {
    String keyword = list.get(j);
    if ($summary.contains(keyword))
    {
      matched = true;
      tempMatches.add(i);
      drools.getWorkingMemory().setGlobal("matches",
          tempMatches);
      break;
    }
  }
}
List<Integer> currentMatches = (List<Integer>)drools
   .getWorkingMemory().getGlobal("matches");

if (currentMatches.size() == keywords.size())
{
  reaction.add("Exactly, that's it doctor!");
}
else if (matched && repeat)
{
  reaction.add("*nods*");
}
else if (matched)
{
  reaction.add("*nods*");
}
else if (repeat)
{
  reaction.add("I believe you already said that.");
}
else
{
  reaction.add("Not really.. Can I talk to my own GP
     ?!");
```

```
  }
end

rule "Finished"
when
  Finished()
then
  FinishedSceneEvent f = new FinishedSceneEvent();
  f.setSceneName("Closing");
  insert(f);
end
```

Appendix C

Walkthrough

In this appendix we give the complete walkthrough of the doctor-patient scenario of SALVE. This is one of the possible ways to walk through the scenario. Each sentence has multiple different ways of saying them. In the walkthrough, the first utterance (the bullets) are always from the player / doctor. The second utterance (the dashes) are always from the agent / patient.

- Good morning.
  - Hello
- My name is Chris Vergeer.
  - Hello Chris Vergeer, a pleasure to meet you. My name is Mark van Asperen
- I am an intern.
  - Ah you seemed quite young to be a doctor.
- I read you are a patient of doctor Aarts.
  - Yes that is my GP. Is he not here today?
- Your doctor is currently busy with another patient.
  - Ah okay.
- I see your last visit was a year ago.
  - That was for a persistent throat infection, if I remember correctly.
- Just need to write down your name and date of birth.
  - So that is Mark van Asperen. I was born on 16-01-1976, doctor.
- How are things going at home?
  - Things are going alright. My wife was here with our youngest a week ago for a respiratory infection.
- I understand you requested an urgent meeting.
  - Yes it's nice I could come so quickly.
- I am taking over some patients of doctor Aarts today.
  - How come?

- Do I understand correctly that this annoys you?
  - Well it isn't great. But whatever.. assistants work here more often.
- After my research, I will share share and discuss my findings with your GP, doctor Aarts.
  - Okay!
- I will ask some questions first about your complaint, followed by a physical examination.
  - Okay!
- Would you like to know or add something?
  - No everything is clear, doctor!
- About the reason for the appointment..?
  - Yeah I have quite the problem, I'm in quite a bit of pain!
- That's not good..
  - No, you can say that again!
- What kind of pain are you feeling?
  - Somewhat smarting pain.
- As if there are needles stinging you?
  - Yeah, sort of.. yeah.
- On a scale from 0 to 10 where 0 means no pain and 10 means the most severe pain imaginable: how severe is your pain?
  - Well I would almost say a 6 or a 7.
- Can you tell me where the pain is exactly?
  - In my throat.
- Is the pain in the front or the back of your throat?
  - In the front of my throat, not the back.
- How deep is the pain?
  - Not very deep. I would say near my vocal cords.
- Can you point to where it hurts?
  - Yeah right here (points to throat)
- Since when do you have this pain?
  - For a while now.
- What do you mean by a while?

  - It has been over a week now.

- Have you ever had this before?
  - No, never had this before!
- Has there been any change in the issue over time?
  - It did become worse the past 2 days.
- Is there an immediate cause for this complaint?
  - No I have not used my voice any different than usual. Have not been out in the wind or eaten spicy foods. I think it's something else.
- What do you think it could be?
  - Well, it could be a cold. My two children also have a cold.
- Is there anything that makes it less painful?
  - No, not really.
- Did you try anything to reduce the pain?
  - I tried throat pastilles, but those things hardly do anything.
- Is there anything that intensifies the pain?
  - If I start screaming haha.
- Are there any other complaints or symptoms?
  - Let's see.. I am also tired..
- How does this fatigue show itself?
  - Well, I feel exhausted and lifeless.
- Is there a reason for that?
  - Of course! Because of my sore throat I can hardly get to sleep at night. I usually fall asleep at like 4 o'clock.
- When did this start?
  - About a week ago.
- So fatigue is also in play.
  - Correct.
- How do you experience the pain?
  - It's very annoying!
- How come?
  - I have a gig with the band that I play in soon and I might have to cancel that because of this!

- You are afraid your performance will be compromised?
  - Yeah. It takes so much rehearsal time. This gig could be our break through. And now it might be compromised..
- That is not good!
  - No, you can say that again!
- Did you think of any causes for the pain?
  - I guess it might just be a common cold.
- Why do you think so?
  - It's going around lately.
- Is it?
  - Yeah, two of my children have a cold as well.
- It is completely understandable that you think of a cold!
  - Exactly!
- Is there any other possible cause for this?
  - If I think about it, I'm considering a lot of things.
- Like what?
  - Maybe I have some kind of problem with my immune system.
- What makes you think that?
  - Well this is the third time this year that I've gotten sick. A cousing of mine had that as well and it turns out he had some kind of immune disorder.
- Yeah I can imagine that left a mark, if it was that close.
  - Exactly!
- How does the pain influence your daily life?
  - It's hard I have had to limit the rehearsals with the band to a minimum.
- That is a problem?
  - Well, I earn my money through these performances and I'm giving a concert in the local theatre at the end of this week!
- You are a singer?
  - Yeah and I really MUST prevent that performance from being canceled.
- How is that pain in the home situation?

  - Well that isn't the problem.

- So it is okay at home I understand?

  - No problem! The household is still going strong. My wife is a real housewife. As long as she isn't sick, everything is fine around the house.

- And what is the problem then?

  - Well, I earn my money through these performances and I'm giving a concert in the local theatre at the end of this week!

- Is there any special reason why specifically this concert must go on?

  - Well yeah, somebody is coming to watch the band who could get us a gig in the United States. That would give us a good chance to REALLY break through!!

- Can you still exercise your hobbies?

  - Yes, my hobby is my music of course! And that is also my job..

- Have you tried anything to reduce the pain?

  - I got some of those throat pastilles from the apothecary, but it only eases the pain for as long as the pastille lasts.

- There was nothing else you could do?

  - No, nothing...!

- Did you try cutting down on the amount of usage of your voice?

  - Yes, I reduced the amount of rehearsals with my band.

- Does your wife offer any support?

  - second

- Can the problem that is caused by the pain be solved within the band?

  - Not really, but she doesn't need to!

- Did you think of what would help you most?

  - I would really appreciate it if you would give me a strong treatment for it

- Is there any particular reason you want a strong treatment?

  - I have to get my voice back in order to do my work.

- You mean because of your performance as a singer?

  - Exactly!

- Well Mister van Asperen, I can imagine that!

  - So could you prescribe that?

- I'll discuss that with your doctor.
  - Thank you!
- I understand you want to get rid of your problem as quickly as possible!
  - Of course! The sooner, the better!
- I will summarize my findings for you.
  - Please do.
- If I understand correctly, you have been having a sore throat for about a week.
  - *nods*
- This is interfering with your night 's rest, leaving you weary.
  - *nods*
- You fear that this might endanger your performance.
  - *nods*
- You would prefer I prevent this by giving you a strong treatment.
  - Exactly, that's it doctor!
- Okay I will have a chat with your GP now.
  - Okay, thank you.

References

[1] *25 Chatbot Startups You Should Know - ventureRadar*. http://blog.ventureradar.com/2016/06/14/25-chatbot-startups-you-should-know/. Accessed: 2017-03-04.

[2] *Alice Bot*. http://www.alicebot.org. Accessed: 2016-05-16.

[3] *Alicebot: Mitsuku wins Loebner Prize 2013*. http://alicebot.blogspot.nl/2013/09/mitsuku-wins-loebner-prize-2013.html. Accessed: 2016-05-16.

[4] Esteban Aliverti. *Mastering Jboss Drools 6*. Packt Publ., 2016.

[5] James Allen, George Ferguson, and Amanda Stent. "An architecture for more realistic conversational systems". In: *Proceedings of the 6th international conference on Intelligent user interfaces*. ACM. 2001, pp. 1–8.

[6] James F Allen et al. "Toward conversational human-computer interaction". In: *AI magazine* 22.4 (2001), p. 27.

[7] Agnese Augello et al. "Humorist bot: Bringing computational humour in a chat-bot system". In: *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*. IEEE. 2008, pp. 703–708.

[8] Pierre Bourdieu. *Outline of a Theory of Practice*. Vol. 16. Cambridge university press, 1977.

[9] Hsiao-fang Cheng. "A Comparison of multiple-choice and open-ended response formats for the assessment of listening proficiency in English". In: *Foreign Language Annals* 37.4 (2004), pp. 544–553.

[10] Herbert H Clark. *Using Language*. Cambridge University Press, 1996.

[11] F Dignum et al. "Situational deliberation; getting to social intelligence". In: *Computational Social Science and Social Computer Science: Two Sides of the Same Coin* (2014).

[12] Virginia Dignum and Frank Dignum. "Contextualized Planning Using Social Practices". In: *Coordination, Organizations, Institutions, and Norms in Agent Systems X*. Springer, 2014, pp. 36–52.

[13] *Drools - Business Rules Management System (Java, Open Source)*. http://www.drools.org. Accessed: 2016-08-18.

[14] *Fable (video game)*. https://en.wikipedia.org/wiki/Fable_(video_game). Accessed: 2016-06-07.

[15] Anthony Giddens. *The constitution of society: Outline of the theory of structuration*. Univ of California Press, 1984.

[16] John O Greene and Brant Raney Burleson. *Handbook of communication and social interaction skills*. Psychology Press, 2003.

[17] Thomas Holtgraves and Tai-Lin Han. "A procedure for studying online conversational processing using a chat bot". In: *Behavior research methods* 39.1 (2007), pp. 156–163.

[18] *Home Page of The Loebner Prize*. http://www.loebner.net/Prizef/loebner-prize.html. Accessed: 2016-05-16.

[19] Torild Jacobsen et al. "Analysis of role-play in medical communication training using a theatrical device the fourth wall". In: *BMC Medical Education* 6.1 (2006), p. 1.

[20] Johan Jeuring et al. "Communicate!—A Serious Game for Communication Skills—". In: *Design for Teaching and Learning in a Networked World*. Springer, 2015, pp. 513–517.

[21] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[22] James Moor. "The Dartmouth College artificial intelligence conference: The next fifty years". In: *Ai Magazine* 27.4 (2006), p. 87.

[23] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005.

[24] Andrew Ortony, Gerald L Clore, and Allan Collins. *The cognitive structure of emotions*. Cambridge university press, 1990.

[25] *Otoscope*. https://en.wikipedia.org/wiki/Otoscope. Accessed: 2016-07-21.

[26] Paul Piwek and Kees Van Deemter. "Towards automated generation of scripted dialogue: some time-honoured strategies". In: *arXiv preprint cs/0312051* (2003).

[27] Mark Proctor et al. "Drools documentation". In: *JBoss* 5.05 (2008), p. 2008.

[28] Andreas Reckwitz. "Toward a theory of social practices a development in culturalist theorizing". In: *European journal of social theory* 5.2 (2002), pp. 243–263.

[29] Theodore R Schatzki. *Site of the social: A philosophical account of the constitution of social life and change*. Penn State Press, 2010.

[30] Theodore R Schatzki. *Social practices: A Wittgensteinian approach to human activity and the social*. Cambridge Univ Press, 1996.

[31] Bas R Steunebrink, Mehdi Dastani, and John-Jules Ch Meyer. "The OCC model revisited". In: *Proc. of the 4th Workshop on Emotion and Computing*. 2009.

[32] *The CMU Pronouncing Dictionary*. http://www.speech.cs.cmu.edu/cgi-bin/cmudict. Accessed: 2016-05-16.

[33] *The History of the Screenplay*. http://thescriptlab.com/feature/category/screenwriting-101/3147-the-history-of-the-screenplay#. Accessed: 2016-06-06.

[34] Alan M Turing. "Computing machinery and intelligence". In: *Mind* 59.236 (1950), pp. 433–460.

[35] Loïs Vanhee, Huib Aldewereld, and Frank Dignum. "Implementing norms?" In: *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 03*. IEEE Computer Society. 2011, pp. 13–16.

[36] *What is AIML?* http://www.alicebot.org/aiml.html. Accessed: 2016-05-16.

[37] Michael J Yedidia et al. "Effect of communications training on medical student performance". In: *Jama* 290.9 (2003), pp. 1157–1165.