Utrecht University

Master Thesis

# Jungian personality in a chatbot

*Author:*
Jappie J. T. Klooster

*Supervisor:*
Dr. Frank Dignum
Dr. Medhi M. Dastani

Faculty of Science
Artificial Intelligence

June 27, 2017

## Abstract

We explore how to add Jungian personality to a chatbot as a process. 'Salve', an existing serious chatbot game, was used as a starting point and closely inspected. We designed personality as a preference for an algorithm rather than value driven decision making, akin to the ideas of Campos [1]. In other words, a personality prefers a way of doing things rather than the content they prefer. With this in mind a strategy was devised for adding personality to the existing chatbot in the 'Salve' serious game, while leaving as much of the original architecture in tact. This caused us to replace the existing AIML based scheme with a novel approach. In this approach, we opened up space to have varied responses to a similar utterance depending on the personality. The Drools rule engine is now the center of deliberation in the chatbot. The new modelling method was also less verbose and more precise. We conclude by demonstrating that this scheme works as expected.

### Keywords

Jung, chatbot, Drools, YAML, AIML, MBTI, salve, communicate!

# Contents

# 1 Introduction

Communication is the foundation of our modern society. Having good communication skills can help individuals in both their professional and personal lives. However training people in communication skills can be difficult. Another party is required to communicate with, and a tutor or teacher has to be there to give feedback.

Serious games can be used to train people with these kind of skills [2]. An example of this is the 'communicate!' game which was developed specifically for medical students. Wherein teachers can create scenarios to let their students practice with communication [3]. The 'communicate!' game was script based, the teacher made a scenario and the student would follow the choices predefined by the teacher. A script based game has of course the weakness that a student can't use creative responses, all possible responses are scripted into the scenario and replying correctly relies on a simple $ABC$ choice.

To deal with this, an alternative serious game called 'Salve' was made, based on the chatbot Alice [4]. To measure how a student performed, a production rule engine was used called Drools. The social practice of a doctor consult was used in this particular instance, in other scenarios other social practices could be used. In this thesis we are interested in extending this work with personalities, where we consider personalities to be a preference for a process rather than content [1]. This is useful because it turns out that the issues most doctors struggle with isn't so much being sensitive, but rather being sensitive to the people who appreciate it [5]. Therefore extending the game so that doctors can train for dealing with different personalities, will help addressing this issue.

In background section 2 we will start by looking into personality theories 2.1, and consider the advantages and disadvantages they have for our use case. Then we'll discuss agents literature in 2.2. followed by the idea of social practices 2.3, and some limited speech act theory 2.4. After which we continue with dialogue systems 2.5, some of which can be considered related work. We continue by taking a close look at the state of the existing 'Salve' software that we're planning to extend in section 2.6. Since the personality topic has become quite popular in recent years some thoughts will be devoted to related work in section 3. With all this in mind, we combine Campos' architecture with dialogue and Jungian personality as a theory in section 4. To make the work in the previous section more concrete, we discuss what architecture was used to implement this in section 5. In section 6 we go into detail why we replaced AIML and with what. Followed by section 7 in which we discuss a scenario 7.1 and how to use the bot 7.2, after which we present the test results 7.3. Finally we conclude in section 8, in which we also discuss some possible issues 8.1 and we present a list of future work 8.2.

# 2 Background

In this chapter we will discuss the work that is the foundation of this thesis. First we will look at personality theories developed by psychology. Then we will look at some to the literature in AI and argue for the methodologies used, and finally we will look at the serious game in its existing form.

## 2.1 Personality theories

A personality is a set of identifiers that can be used with reasonable consistency to predict behavior [6]. What we want from this model is a guideline of implementation for the program, that is to say, the more the theory says about internal workings of a person the better it is. We want the model to be realistic of course, but we also want it to be implementable. This is where we have a conflict of interest with the field of psychology since they do not necessarily care about implementation details.

This conflict of interest can be seen for example in [7], where a criteria of personality is that it should be stable and coherent. However this is a poor software specification since there is no unit of measurement (how long should it be stable, and what range is acceptably stable), but for psychology it is a good definition, because a human can determine out of context what these things are.

The field of psychology has been somewhat active in trying to model human personality [8]. Several frameworks have been developed to figure out people's personality and what this in turn would mean for their lives. We are interested in two ways in existing personality theories:

1. Accuracy, if a personality theory does not fit the reality at all it won't help anyone in the serious game.

2. Ease of implementation. If the personality theory is too hard (or impossible) to implement in the serious game then we can't use it.

The field of psychology is very interested in the first requirement. However the second requirement not so much. Therefore our first job will be to list existing psychology personality frameworks, and filter out those that are unfeasible to implement.

### 2.1.1 The big five

The first framework we'll discuss is called the big five. The term big five was first coined in 1981 by Goldberg [9]. The big five were not big because of their intrinsic greatness, but rather to emphasize how broad these factors were.

This framework was not really invented, but rather discovered trough lexical analysis by for example Tupes [10]. Although the labels used were different, they conveyed the same idea as the big five model used now. The methodology used is something which is called factor analysis[1]. Factor analysis is a statistical methodology that tries to find underlying hidden variables. This methodology has become widely used in psychology [12].

[1] In the paper the term 'varimax rotational program' is used, but if we look this term in Wikipedia, we can see the result is called factor analysis [11]

The data Tupes used is from Cattell [13] and several others. Cattell used a rating scheme, where a trait was introduced and all test subjects then had to rate all other test subjects as average, below or above average for that specific trait. Subjects were also required to select two extreme trait ratings (max and min) in the subject group. These traits in the test were based on the *personality sphere* concept which tried to cover the entire surface of personality by providing many small trait areas. Examples of the traits are: 'Attention getting vs Self sufficient', or 'Assertive vs Submissive'.

In the beginning of the 1990's there were many ways to measure personality that didn't agree with each other. For example at Berkeley, Block used a 2 dimensional ego-resilience and ego-control method [14], whereas Gough measured folk concepts such as self-control, well-being and tolerance [15]. Personality researchers hoped that they would be the one to discover a structure that would then be adopted by other researchers [16].

The goal of the big five was not to present a new structure that convinced others to use it, but rather to provide a taxonomy that all psychologist could agree upon. Since the big five was so broad (because of the statistical methods used), this worked. Therefore the researchers could keep on exploring there niche with their proffered structure, but once they would present their work they could use the big five to communicate clearly what their research meant without having to redefining the words every time [17].

The big five as in the OCEAN definition has the following units of measurement:

- Openness or originality, if you score high on this you enjoy learning new things just for the sake of learning. If you score low then you don't enjoy this
- Conciseness, how tidy you are, if you score high the dishes don't stack up in the sink.
- Extroversion, a high score indicates you enjoy leading the conversation and you'll speak up when you disagree with someone.
- Agreeableness or altruism, a low score would indicate that you don't want to share and generally don't trust people.
- Neuroticism or nervousness, a high score indicates that you like to brag and get upset when someone is angry at you.

The big five has been extensively tested and the result has been replicated in multiple studies [18]. One can measure big five score trough a test called the NEO-PI, or the NEO-FFI. The FFI variant is shorter but less precise [19].

Although these terms may provide a great taxonomy, it does not have any theoretical foundation [20]. This means it becomes difficult to speak about implementation. To make this more clear we use a thought experiment: Lets say you have a score of 0.8 for Neuroticism, how does this influence my decision for selecting action $a$ or $b$? Now you could say, use a mixed strategy where in you choose 80% of the time the neurotic typical neurotic approach. Then we need a valuation function to decide which of the two actions is more neurotic. But once we've done this we still haven't taken into account any of the other factors. Solving this is a non-trivial endeavor.

There are some existing solutions in which OCEAN is implemented, for example Allbeck [21] used it as a mapping to the EMOTE system, whereas [22] used the OCEAN values as a low level mapping in steering behaviors and finally [23] used the values for action selection in a dialogue, but extended the descriptions of OCEAN with IPIP with an entire chapter devoted to explaining this. Although these implementations are based on the same OCEAN model, the influence of it has starkly different effects on their respective implementations. Since each of them decided to change the OCEAN model in some kind of way we can conclude that although OCEAN is good for discussing the psyche, it is incomplete for a software specification role.

### 2.1.2 Personality types

To address the big five's issue of having no theoretical foundation we'll inspect the idea of personality types. We begin with the theoretical foundation proposed by the grandfather of personality research, Carl Jung. After which we'll look at a theoretical evolution proposed by Myers and Myers-Brigs, which also introduced a structured method of measuring types. Then we'll discuss some critique on this method. With this criticism in mind we consider some alternatives to the MBTI that have been proposed afterwards.

**Jung's theory of psychological types** Jung describes several concepts, firstly each person has two attitudes: *Introversion* and *extroversion*. Extroversion means dealing with the outside world and therefore is called objective (or observable). Introversion is the world inside a person, and therefore is subjective, or private. These attitudes are mutually exclusive, you can't do introversion and extroversion at the same time. For example if you're day dreaming, you're not paying attention to your surroundings. A person who spends most of his time in the introversion attitude is called an *introvert*, conversely someone who spends most of his time in the extroversion attitude is called an *extrovert*. One is however never totally an introvert or extrovert, an introvert can still have extrovert moments and vice versa. It should also be noted that the unconsciousness according to Jung is flipped in attitude. [24]

Then there are four functions. The first two functions are called the *rational functions* because they act as a method of making judgements. *Thinking* is a function that connects ideas with each other to arrive at generalizations or conclusions. *Feeling* evaluates ideas by determining if they are good or bad, pleasant or unpleasant, beautiful or ugly. Note that this is *not* the same as being emotional, although you can be emotional and use this function. The *irrational functions* are called this because they require no reasoning. *Sensation* is sense perception created by the stimulation of the senses, it can always be rooted to a sense, such as "I see a balloon" or "I feel hungry". *Intuition* is like a sensation but it's not produced by a sense. Therefore it has no origin in the same way as sensation has, and often is explained as "just a hunch" or "I feel it in my bones". [25, 26]

To use these functions they have to be combined with attitudes, producing *function attitudes*. Therefore a person will never be of a thinking type, but rather either a thinking introvert or thinking extrovert. [27] We can now imagine what this means, an extroverted thinker will for example make judgements about the real world, and therefore be more like a natural scientist or biology

researcher, where they would study natural objects and behaviors. An introverted thinker will make judgement about ideas in his mind, and therefore will be an excellent philosopher, or mathematician, where consistency of the internal reasoning process is important.

Let $\mathcal{J}$ denote the set of all possible Jungian function attitudes such that:

$$\mathcal{J} = \{T_e, T_i, F_e, F_i, S_e, S_i, N_e, N_i\}$$

Where

- $T_e$ stands for extroverted thinking, which is thinking about objects in the real world. This is thinking with a goal, a problem to solve, to check weather certain laws are upheld, or a system to check. As said before a typical example of $T_e$ based reasoning would be a biologist studying natural behavior.

- $T_i$ stands for introverted thinking, this kind of thinking could be called deductive, it tries to construct a framework to explain the world. This is consistent reasoning based on internal believes, which does not necessarily solve a problem. A typical example of $T_i$ based reasoning is a mathematician creating or combining new mathematical structures with help of axiomatic logic.

- $F_e$ stands for extroverted feeling, where objective or external criteria is used to judge, for example something is beautiful or ugly. Established standards may be used to decide this and therefore it's a conservative function. Decisions are based on interpersonal and cultural values. A typical example of $F_e$ based reasoning is about fashion and fads. Deciding what is fashionable at the moment is an $F_e$ based process. A typical profession would be working at a clothes shop, where the knowledge of the latest trends is crucial.

- $F_i$ stands for introverted feeling, decisions based on personal values and believes. People who have this as dominant function attitude could be characterized by 'still waters run deep'. A typical profession for this type is in counseling or health care, because empathy comes rather natural to them [28].

- $S_e$ stands for extroverted sensing, Act on concrete data from the here and now. Then lets it go. People of this type are often realistic and practical. A typical profession driver of heavy machinery or athlete [29], because living in the moment is most important for those professions, this comes natural to $S_e$ based personalities.

- $S_i$ stands for introverted sensing, acts on concrete data from memories and passed experience. A possible profession for the people with $S_i$ as dominant function is in quality assurance, where the perfect model in their mind can be easily compared to the product in question [30].

- $N_e$ stands for extroverted intuition, try to find possibilities in every situation. Extroverted intuition can be very good entrepreneurs, seeing ideas in almost every situation, this also makes them very inspiring leaders because they are very excited about their ideas [31].

- $N_i$ stands for introverted intuition. Looks for new possibilities in ideas. A typical occupation of this type is artist or visionary [32], this is because connecting ideas with each other comes natural to this type. However just like the typical artist it may not always be understood why by his peers or even himself.

Another important concept is the idea of the *principal* and *auxiliary* function [33]. The principal function is the one that is most preferred. The auxiliary renders its services to the principal function, however this function cannot be the opposite of the principal. So if *Feeling* is the principal function than thinking cannot be the auxiliary. This is also true for the irrational functions.

**MBTI** The Meyer brigs type indicator is based upon Carl Jung's theory of personality types. However it brings two important changes, first of all the way of measuring personality type is changed. It uses a structured approach rather than Carl Jung's projective approach. The responses to items are finite and therefore can be deduced based on theory. In contrast to Jung's technique where he used open ended answering with word associations [34]. Then there is the introduction of an extra index used to order function attitudes [35]. Which is either a $J$ for judging (rational in Jung terms) or a $P$ for perceiving (irrational in Jung terms). This dimension indicates together with the $I/E$ dimension which function attitude is dominant and which is auxiliary.

Once completed with the MBTI you'll get character string as outcome, for example 'INTJ'. This label tells you indirectly which of Carl Jung's functions is dominant, auxiliary, tertiary and inferior [36]. In other words it provides a sequence of preferences [37]. In case of INTJ it would be:

$$N_i > T_e > F_i > S_e$$

So the most preferred function to be used by someone of type INTJ would be $N_i$, then $T_e$ and so forth. These are the same functions as Jung used, the MBTI just imposed an order on them [36, 38]. How much preference there is for a function is not encoded in MBTI, just an order of preference. An ENTJ would be similar to INTJ but with a different order:

$$T_e > N_i > S_e > F_i$$

With this definition the interplay of the judging/perceiving dimension becomes more obvious if we look at INTP:

$$T_i > N_e > S_i > F_e$$

It's similar to an ENTJ, but the attitudes have flipped.

A possible grouping of the sixteen type exists using the middle letters:

$$\{NT, ST, NF, SF\}$$

This grouping goes under the rationale that the first two functions only differ in either attitude, order or both.

Before continuing we would like to say a word about a popular interpretation of MBTI which is based on Keirsey's book 'Please understand me', and later 'Please understand me II'. In this interpretation the sixteen types are also placed

in general groups of four but here the $ST$ and $SF$ distinction is replaced by $SJ$ and $SP$ [39]. It turns out however that Keirsey invented this distinction because 'He thought it made sense to group them this way' [40]. In doing this he rejected the work of Jung and also that of cognitive functions. Which is problematic because the theory he presented then does not make any theoretical sense. Therefore Kersey's MBTI will not be used in this thesis.

The MBTI is extremely popular in a sub field called Organizational Development (OD) [41]. But it has gotten some heavy criticism from the field of psychology.

MBTI has always used a continuous scoring system in the results. However the creators insist that type is enough for making assessment judgments. Since MBTI reduces the test scores to type, it is expected that most of the population would fall into either proposed dimensions. For example $I$ or $E$. This is called a bimodal distribution. However [42] suggests this is not the case, but this could be the result of the scores being bidirectional [43]. In an extended investigation [44] into wether Jungian constructs are truly categorical suggested however that this was maybe not the case and a continuous scale for assessment judgements are required.

In [45] the MBTI is put trough a method called factor analysis. This is the same technique where OCEAN is based upon (see section 2.1.1). With this technique the desired outcome is that there are four question clusters (or factors), one for each dimension. They should be independent, a question that influences $I/E$ score should not influence $S/N$, and finally we expect the factors to indicate differences between individuals, random questions won't do that. However the study indicated that the MBTI had more than 4 factors (6), they explain the first extra factor as questions that assessed people being 'unconditional positive', but could not explain the other extra factor. Something else of note was that there were questions doing no discrimination at all (not being scored).

Reliability indicates how often the same result will come out of the test, for example if you take the MBTI a 100 times you may be classified the same type for 70 times, which would be an indication it has a reliability of around 70%. But in psychology another aspect is important, namely the interval in between which the tests are taken, if for example two tests produce starkly different results but a long time has passed between them it's not considered a big issue. In [46] it is suggested that after a period of 5 weeks 50% of the participants changed in score. However one should take into consideration that after taking the test a first time people could consciously decide to change their opinion because they think it's more desirable to have a different type. Jung said that type is decided very early on in life [47], so reliable scoring is important.

**PPSDQ** The PPSDQ keeps basically the same theory as MBTI [48, 49], but uses a different measuring method. Instead of forced questions it uses a word-pair checklist for $I/E, S/N$ and $T/F$ scales, and for the $J/P$ scale self describing sentences are used [50]. An example of a word pair checklist can be found in table 1. The word pairs themselves were obtained by prescribing an exploratory test(s) to a sample in which the proto PPSDQ was submitted and also the MBTI itself, factor analyses was used to determine correlation, this is done in [51]. The optimal amount of points (options to choose from) presented in such a test is a subject for debate. Common sense would suggest that more points would

give more precision, but in [52] it is suggested that reliability and validity do not increase with more points. In [53] however they state the importance of an available midpoint. The 5 point choice format in the PPSDQ is not motivated.

| Word | | | | | | Word |
|---|---|---|---|---|---|---|
| Empathy | 1 | 2 | 3 | 4 | 5 | Logic |
| Dispassionate | 1 | 2 | 3 | 4 | 5 | Emotional |

Table 1: An example of a word pair checklist, where the test taker should choose the word that he identifies most with.

The result of the PPSDQ would look something like: $I-30, N-20, T-80, J-60$, with a scale of 0 to 100. This means the tendency for introversion would be about 30, and similarly for the other dimensions. Note that $I-30$ is the same as $E-70$, introversion and extroversion being on opposite ends of the same scale. As an application, we can for example make a preference sequence, where higher valued functions come first, or create a mixed strategy, where we select functions based upon probability.

The PPSDQ is measuring the same thing as MBTI but lacks the criticisms of MBTI. The reliability is for example between 90% to 95% with a delay of two weeks. The internal consistency was also measured which proved to be better than MBTI but there was still a dependency between $S/N$ and $P/J$ which remains unexplained [48]. The PPSDQ is internally the most consistent of the discussed alternatives (excluding OCEAN) [54].

**SL-TDI** SL-TDI measures functions by presenting 20 situations and then giving subjects possible actions which correlate with the functions. The subjects then have to indicate how likely it is that they would choose that particular action [55].

It becomes rather straight forward to make a function preference of the measurement of SL-TDI since the question directly measures the Jungian functions. A possible personality type therefore would be:

$$S_i \geq T_i \geq S_e \geq F_e \geq N_i \geq T_e \geq N_e \geq F_i$$

To determine the preference we just used the observed value in the test. Since every situation offers a choice for each function with a 5 point value there is no need for normalization.

This denotation is much less strict than the MBTI or PPSDQ since it does not force alternating attitudes or pairing of rational/irrational functions in the preference. Therefore the amount of personality types SL-TDI supports drastically exceeds that of the PPSDQ. In other words, there always exists a mapping from PPSDQ to SL-TDI, but not always from SL-TDI to PPSDQ. The reason for doing this is because there is experimental evidence that there exist personalities outside of the structure originally imposed by MBTI and the subsequent PPSDQ [56].

### 2.1.3 Comparison of theories

To re-iterate, we are interested in a framework that is realistic, and easy to implement. The Big Five falls short on the easy to implement, there is no

underlying theoretical framework to support it [20], therefore we cannot base our implementation on anything except our own interpretation.

The MBTI has been criticized a lot from the field of psychology, but it does have a solid theoretical foundation. There is some relation between the big five and MBTI [57]. Therefore it's somewhat realistic, but quite easy to implement.

Both of the alternatives of MBTI use a continuous scale and have a high correlation with the big five [58]. This means is that they are measuring something which is also measured by the big five in some way.

The PPSDQ is based on the same theory as MBTI, but with scaled type letters. To convert the type to function attitudes some extra work has to be done, namely calculate their respective probabilities. To decide which function attitude to use some kind of mixed strategy has to be used. The PPSDQ is more realistic, but at the cost of being more difficult to implement.

The SL-TDI is even harder to implement than the PPSDQ because the function attitudes no longer have to alternate. This either means that functions are independent (thereby rejecting some of Jung's work), or that they have to work in some kind of combination. If they work in some kind of combination and we have the following preference:

$$T_e > T_i > S_i > N_i > F_e > N_e > S_e > F_i$$

We select the first function to work with, but it requires some information, which we can only get from an irrational function. So what do we do now? Select $S_i$, thereby skipping $T_i$, or select $T_i$ and let it decide to select $S_i$, but this would basically give $T_i$ censorship rights. This is difficult to answer therefore it is a lot more difficult to implement than PPSDQ. Since SL-TDI drops an assumption, which is shown with experimental evidence to be false [56], we can say SL-TDI's theory is most realistic, but this comes at the cost of being even more difficult to implement.

Therefore our preference for implementation is the following:

$$\text{MBTI} > \text{PPSDQ} > \text{SL-TDI} > \text{OCEAN}$$

There is another hidden reasoning behind this, the work of PPSDQ can built on that of MBTI, and that of SL-TDI can build on that of PPSDQ. OCEAN lacks theory and builds on statistics, however since SL-TDI and especially PPSDQ have a statistical relationship with OCEAN [54], Jungian theory can be used quite realistically with an eventual statistical mapping mapping back to OCEAN.

## 2.2 Agents

In the literature there is little consensus on what exactly an agent is, however there is a general consensus that an agent is *autonomous* [59]. To make this more clear we'll use Wooldridges' definition:

> An *agent* is a computer system that is *situated* in some *environment* and that is capable of *autonomous action* in this environment in order to meet its delegated objectives.

In another older definition [60] Wooldridge highlights *autonomy*, *social ability*, *reactivity*, and *pro activity*. Where autonomy means that no human intervention is required, social ability means it can talk to other agents, reactivity is that it can reply on input and pro activity means that it can show behavior while not reacting to something. However he later continues on with a stronger claim: An agent is a piece of software that uses concepts which are attributed to humans, such as believes desires and intentions.

This is the reason why we can't call any program an agent. For example an operating system kernel is autonomous (a user would never interact with it), social (can do networking), reactive (it will comply to hardware interprets for example) and proactive (a process hogging to much memory will be killed without the process asking for it). However we won't call a kernel an agent because it doesn't even come close to having believes, desires or intentions.

Something to keep in mind is that there are three 'branches' of agent research [60]. The first one is *agent theory* in which *specifications* and methods of specifications are developed. They ask what are agents and what are they ought to do and how do we tell them that, we describe some in section 2.2.3. Then there are the *agent architectures*, these address questions of how to implement the specifications written by the theorists. Although we already got an architecture described in section 2.6, we will explore some more in section 2.2.2. To show some comparable architectures to our own. Finally there are the *agent languages*, which ask the question how to write agent programs. This again is mostly predetermined for us, but we briefly mention some in section 2.2.4, to juxtapose with our approach.

### 2.2.1 Belief desires and intentions

The belief desire intention model of human practical reasoning was first introduced by Bratman [61]. It is based upon a 'common sense' framework of human reasoning.

The idea of BDI is that an agent has believes, these can be anything, such as I believe the grass is green, or I believe the keys are on the table. Note that we never speak about facts, an agent can believe something to be a fact, but that doesn't make it a fact. Desires are special kind of believes that give agents a reason to be, they may also be called goals. Intentions are (partial) plans to make a desire come to fruition. How to formalize this properly turns out to be a hard question, which is analyzed in the following section 2.2.3.

A number of reasons have been stated to use this methodology. The foremost is to make agent orientated systems less expensive in maintenance, verification and construction according to Rao and Georgeff [62]. However they don't cite a source for this.

Another paper argues in favour of agent orientated design [63]. It has the following major arguments: It is effective to divide a complex problem domain into several smaller problems, abstracting in an agent orientated way is more 'natural', and complex systems dependencies and interactions can be easily modeled.

### 2.2.2 Intelligent virtual agents

Intelligent virtual agents are systems that emulate characters, that not just move but have human like abilities [64]. Because of complex cognition and

planning mechanisms they are able to deal with dynamic social environments autonomously.

We can consider the Fatima architecture [65] as an intelligent virtual agent architecture. In this the OCC model [66] was used to define and track emotions in their respective agents. They also defined 'personalities' trough value based limits on emotions, decay rate variances, goals, reaction rules and action tendencies. These don't follow the theory discussed in section 2.1. Note that the Fatima architecture is an extension upon BDI [67].

We can consider the architecture of the 'Salve' game discussed in section 2.6.2 already a virtual agent architecture. It for example also has an emotions module which also is grounded in OCC [4]. So it is only natural to say the architecture discussed in section 5, is also an intelligent virtual agent architecture. This thesis did not try to unify OCC based emotions with personality, moving Drools to the core of deliberation could make this easier however (see section 2.2.4). More extended use of social practice, and the recombination of OCC could in future work lead to an agent that can chat much more naturally than just another Alice bot.

### 2.2.3 Logic of BDI

Logic of BDI is an attempt to formalize how agents behave. One of the first formalization of Bratman's theory was that of Cohen and Levesque [68]. It was based on linear time logic and used operators for actions and modalities for goals and beliefs [69]. It also used a tiered formalism, with at the bottom belief goals and actions which provided the basis for the higher achievement and persistent goals and intentions to do and be. Rao and Georgeff introduced a different formalism that used branching time logic. They use modal operators for belief desires and intentions and then put constraints on them to make interactions meaning full [69]. Therefore this formalism is much closer to that of Bratman [70]. Finally there is the KARO formalism which is based on dynamic logic. This is the logic of actions and computation. They extend this logic with epistemics to add believes to it [69].

### 2.2.4 Drools

If JADE [71], and 2APL [72] are agent orientated programming languages, then Drools can be seen as a more low level variant. Things such as goals and ontology are not predefined in Drools but there exists a concept of rule matching similar to 2APL. Drools is called a production rule system, which is based around the RETE algorithm [73]. A good example in which Drools is used is the expert system called OptaPlanner [74], which is a constraint satisfaction solver trough heuristics by using Drools.

Drools consists of three major concepts. First of all there is the data model, which are just java classes. This data model is called the fact base. Then we have the rule queries, or left hand side. These indicate when a rule should be executed by analyzing the fact base. Finally there is the right hand side, which is java code that gets executed if the left hand side becomes true. This code can modify the facts, or interlope with outside java code trough global variables. Also note that Drools is Turing complete [75]. An example of a drool rule can be seen in listing 1.

```
rule "Create default reply"
when
    $symbol_database:SymbolDatabase()
then
    log.info(drools.getRule().getName());
    delete($symbol_database);
    insert(new DefaultReply($symbol_database.get("nonsense").orElseThrow(
        ()->new RuntimeException("I can't find nonsense anywhere :s")
    )));
end
```

Listing 1: Example of drool rule

With listing 1 we can also see the difference between facts and globals. The facts are used as values to execute Drools upon. In the example this is the `SymbolDatabase`. Globals are interactions with objects that live outside of the rule engine. Such as the logging object `log` in the example.

An interesting difference between traditional BDI model and Drools is that Drools speak about facts. In section 2.2.1 that we never speak about facts, however drool does call the main data model the fact base. Drools is of course not a BDI agent programming language and does not need to keep to the established BDI taxonomy.

### 2.2.5 BDI + Personality

There have been several works that attempted to combine BDI with personality theory. In [76] emotion and personality is taken together and modelled formally. Similarly to the Fatima architecture there is no personality research cited in this work, just research on emotions. However this formal model could be useful regardless of the lack of personality theory, especially the observation that there are only three possible transformations as the result of emotions: Transformations of actions space, transformations of the utility function and transformations of probabilities of state. We mention this because we do transformations of action space (irrational functions) and utility functions ($F_e$ modifies itself) in section 4.4.2. The transformations of state idea may seem foreign to us: Personality never changes in our architecture. However we do have the believes $B$, which could be seen as the state of mind of an agent. Therefore we do this transformations whenever we change the believe base.

In [77] the personality model of Millon was used, they chose to interpret it as a value based personality scheme. Where the values would indicate probability of action selection and quality of behavior. If an agent would get several tasks the one he selects depends on his personality values and the quality of execution also depends on personality. Tasks could align or not align with personality depending on the task. A drawback to such an approach is the necessity of mapping values of the personality to the required actions. We do this to some extend in the symbol graph with perlocutionary values for example, but this is only necessary when certain personalities need to follow different routes (see section 7.2.3), besides the irrational functions do something completely different.

In [1] Campos presents a methodology for adding personality to BDI agents. What is novel about his work is that rather than presenting personality as value driven, it emerge from the process an agent prefers. So personality defines the reasoning approach an agent will use according to Campos. This was called *process orientated* rather than contend orientated. [1] For example in their in-

terpretation of MBTI a sensing agent would make a plan before hand in complete detail (strict evaluation [78]), whereas an intuitive agent would continue planning as the situation demanded from the agent (lazy evaluation [79]). Thinking agents would base their decision process upon their own believes whereas feeling agents would consider what other agents want. In our model we conceptualize the Jungian functions also as a process. We comment more on this in section 4.

## 2.3 Social practice

In [80], practice theory is described as an example of culture theory, from this we can deduce a reason why the study of such theory would be relevant: It can help us explain context in for example dialogue, trough expectations norms and social effects. In contrast to more classical models such as the 'homo economicus' where self interest and goals are most important, and 'homo sociologicus' in which group values are most important. Both these classical models ignore the unconsciousness layer of knowledge humans of same cultures share. Using the social practice model that doesn't ignore this layer, could lead to a more 'natural' conversation with a chatbot.

In [81] it is stated that the research in activity theory led to the development of social practices. It was Karl Marx who thought of the 'roots' of activity theory [82], Activity theory tries to bridge the gap between a single actor and the system it resides in trough the activity in progress [83]. Another way of describing activity in this sense is 'a way of doing things'. A problem with this model however was. How do cultures move activities from the collective towards the individual [81]? Social practices were therefore introduced to make the notion of activity more concrete.

An early adoption of social practice can be found in [84], where it was used to analyze the spread of Nordic Walking. In his analyses he uses the following overarching concepts to analyze the practice:

1. *Material*, which is just stuff in the real world. Such as cars, lamps etc.

2. *Meanings*, which covers issues that are relevant to the material and/or the practice. Think of health, price or even emotions. In [84] meanings and images is used interchangeably, however in [85] it's labeled as just meanings. For clarity we will be using the word *Meanings* since it's more descriptive.

3. *Competence*, to participate in the practice of cycling, one needs to be able to ride a bike. These abilities is what competence encompasses.

In [86] a model of social practices for agents was developed. This model is extended specifically to allow software agents to use it. In this model *physical context* describes the physical environment, it contains resources, places and actors. Note that resources is equivalent to material from the model used by [84, 85]. *Social context* contains a social interpretation, roles and norms. In the previous model this was all part of *Meanings*. *Activities* are the normal activities in the social practice, in Nordic walking this can be for example talking with your partner, or stopping to get a stone out of your shoe. They don't all need to be performed, but are there just as options. This is the first construct that wasn't covered by the other model. *Plan patterns* is a default that is assumed

for all ways the social practice is used. They are concerned with order of activities, certain activities have to happen before other activities. An example of a doctor appointment plan pattern can be seen in figure 1. If you go to the doctor the first thing you do is some kind of greeting. Then the doctor goes onto data gathering and diagnoses mode until he figured out what's wrong. After which he will tell in the finishing phase what to do about it. Now what these phases entail is not clear at all. Finishing may for example contain the prescription of medicine, or an appointment to go to the hospital. However plan patterns do not describe such an implementation, and only constraints on eventual concrete plans. These constraints can be either very loose such as described above, or in certain cases very tight. For example it may be established a doctor can only end the conversation if they asked their patient if they understood everything. This still isn't a concrete plan, since how this is asked isn't described. The plan pattern construct wasn't represented in the previous model either. *Meaning* in this model is solely related to the social effects of activities, and finally *Competences* is the same as in the previous model.

The interest for this model comes from the potential heuristic use of social practices. Once in a particular situation that fits for a social practice the amount of reasoning can be sped up by having actions and their preconditions be grouped under that social practice, if no preconditions match an agent could consider trying other social practices he knows, or ask its peers for more information.

The social practice theory in this thesis should be considered as a *foundation* rather than a separate element. Potentially it could give the notion of culture or even common sense to agents. In this thesis we are interested in implementing personality for a serious game in a single social practice. So right now the social practice just gives an ordered overview in what domain our program should work. We can formulate the social practice that is relevant for this thesis in the following manner:

- Practice name: Doctor appointment
- *Physical context*,
  - Resources: Computer, chair, diagnostic tools..
  - Places: waiting room, doctor's office...
  - Actors: doctor, patient, assistant, ...
- *Social context*,
  - Roles: Doctor, Patient...
  - Norms: doctor is polite, patient is polite, doctor is inquisitive
  - Social interpretation: Can sit on chair, cannot sit on table.
- *Activities*, share information, do diagnostics, minor treatments, prescribing drugs...
- *Plan patterns*, see figure 1.
- *Social meaning*, awkwardness, gratitude, ...
- *Competences*, Give injection, empathetic talk

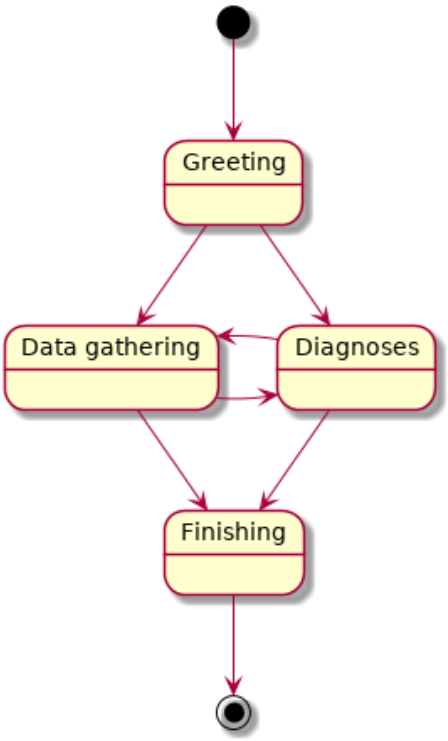


Figure 1: Plan pattern example

We can imagine personality should have $a$ influence on social practice selection and of course plan influence. As far as the authors are aware however, there hasn't been any prior work on this subject, but we can speculate for example that when considering physical context someone that is domination by a sensing extroverted $S_e$ function attitude would check all facts more rigorously than someone dominated by an introverted intuition $N_i$ function attitude.

If the social practices are defined more formally they could be used in a bigger system such as in [87] and [88].

## 2.4 Speech act theory

Since a large part of this thesis is about communication we will give here a brief overview of speech act theory. There are three levels at which speech acts can be analyzed according to [89]. *Locutionary* acts simply convey information form the speaker to the listener. All speech acts do this, as long as they carry meaning. *Illocutionary* acts are the speech acts that do something by saying it. It captures the intend of the speaker. This includes giving orders or uttering a warning. *Perlocutionary* acts are the acts that bring an effect to the hearer, such as scaring or saddening.

There are some basic assumptions of conversation, commonly described as the *rules of conversation* developed by Grice [89]. Human communication happens on the assumption that both parties want to be clear to each other, even when other motivations apply. This is called the *cooperation principle*. To accomplish this shared goal the Grice's maxims [90] are used: *Quantity* has to do with the amount of information transferred in a single utterance, a human wants to transfer just enough to get the right meaning across. *Quality* is the assumption where people will say things they believe to be true. *Relation* states that the things uttered should be relevant to the subject being discussed. *Manner* is about being as brief and clear as possible while avoiding ambiguity and being orderly.

## 2.5 Dialogue systems

Dialogue systems are the systems that try to analyze how dialogue works. This is a sub field of AI that tries to combine linguistics with computer science.

First of all are of course the chatbot systems, which are based upon case based reasoning. A good example of this is the Alice bot [91]. These are mostly reactive systems that use pattern matching rules paired with 'good' responses, sometimes with conditions to allow for more variety. Another example of such a system is Eliza bot which is described in [92], where they also added personality to the bot with the OCEAN model.

Traum [93] describes the information state approach for dialogues. The approach Traum proposes is modeling:

- Informal components, which aren't part of the model but are just there. This can include domain knowledge for example.
- Formal representations, which are data structures.
- Dialogue moves, which entail the set of possible utterances to make.
- Update rules, that allow or prohibit the taking of certain moves.

- Update strategy, to decide what rules to apply at a particular point.

The dialogue is the information state itself [94]. This is an extremely general way of describing a dialogue system.

Both Alice and Eliza fit in this system. Alice for example provides dialogue moves trough AIML and the update strategy is simple pattern matching. You could consider topic tags to be an update rule. The formal representation is then also AIML itself. A similar mapping can be made for Eliza.

In [95] a BDI based methodology is proposed to handle dialogue between a user and an agent. However we want to point out that this solution fits into the rough model Traum sketched. So we could say its a information state approach too.

An interesting paper on dialogue modeling can be found in [96]. What is interesting is that they treat having multiple options available in their implementation (see 3.3 in the referenced paper). This is similar to what we present in section 4.3.2. Although their solution is quite different, rules were made to select according to a single strategy, whereas we saw it as an opportunity to make composable strategies. This is of course an information state approach too.

## 2.6 Salve

This chapter describes the game we inherited from our predecessors. We have to discuss precisely what they did for two reasons:

1. To help understand the design constraints we work under
2. To distinct our changes from theirs'

There have been several distinct versions of the 'communicate!' game. The first version was a web based game, with a scenario editor. [3] However it had some drawbacks, for example each dialog was scripted by the teacher and the answers the student could give were specified by the teacher. This made practicing on it somewhat unrealistic. In this case practicing would mean memorising what button to click rather than to figure out what to say.

To address this issue a new implementation was made. This version was based around the idea of a chatbot, which allowed users to give open answers rather than selecting buttons. The Alice chatbot was used as a foundation and the AIML language was extended to allow emotional reactions of the agent. This new language was called S-AIML [88].

A specific scenario was created for doctor/patient interaction [87]. The game in this version also has the ability to judge the skills practiced [4], such as following certain protocols (politeness, medical standards), and empathy.

There is a difference between the architecture in the published papers and the source code received. This is because the source code is actively being worked on, whereas the papers are snapshots of the source code at the time of publishing. An example of such a difference can be seen if we take [4] in consideration, the judgement of these practices was for example encoded within the S-AIML language, however in the source code AIML has taken a step back. It is only used for text processing and not deliberation (which is now being taken over by Drools as discussed in 2.6.3). Section 2.6.1 and 2.6.2, are based upon

the published papers, however for sections 2.6.3 and 2.6.4, we will be using the source code as a reference when discussing the existing work because it is more relevant.

### 2.6.1 Functionality

There are two major functionality perspectives to consider, that of the student, and that of the teacher. We will consider these in separate subsections since in game they don't interact.

**Student usage** For a student to use the application he has to first start a client. He can now choose to start a new game. There are options to list existing games but these have not been completed. Once in game the user enters a screen as can be seen in 2:


Figure 2: Client view

From here the student can start practicing, the game will track his progress on the server.

**Teacher usage** For the teacher there is no client right now. The way a teacher can setup a scenario is trough modifying AIML and drool files. The teacher probably needs an expert to do this because to load these one needs to do a build. Which can be quite difficult for the first time, as seen in appendix F.

### 2.6.2 Abstract architecture

An abstract architecture was already in place and described very well by [4]. This can be seen in figure 3, which was directly taken from [4].

The `Interaction` module handles user interaction, where the `GUI` can show the dialogue and the mood of the agent. The `Dialogue` module inside it however handles low level string interpretation with help of AIML (see section 2.6.4), this basically works trough string matching. Note that although represented in the

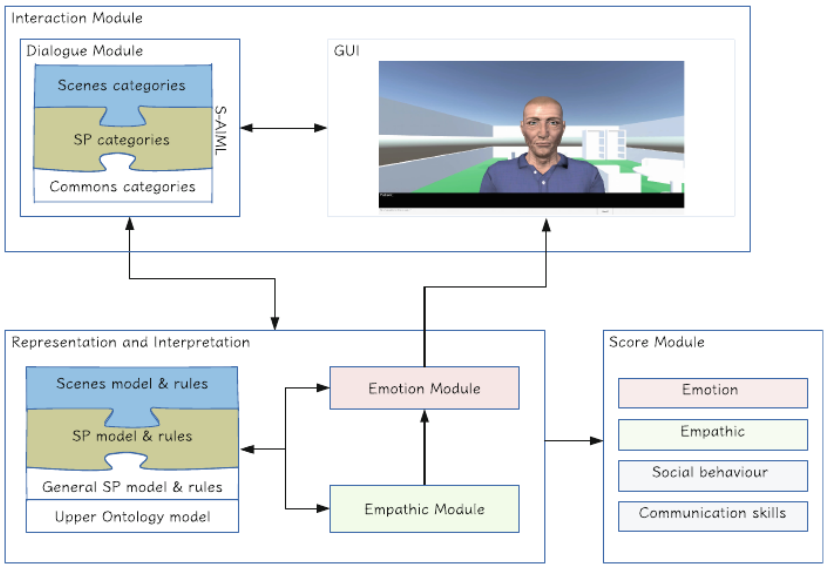


Figure 3: Abstract architecture as described by [4]

abstract architecture as the same module, the `GUI` resides in the implementation on the client side whereas the `Dialogue` module resides on the server.

The `Dialogue` module calls directly the `Representation and interpretation` module, with help of specialized tags (see section 2.6.4) information can be inserted in the `Representation and interpretation` module.

Both the `Representation and interpretation` module and the `Score` module use drools to do their respective tasks. The only real separation in implementation is trough directory and file structure, at runtime there is little distinction. The only other thing of note is the direct connection between the `Emotion` module and the `GUI`, this is done because the `Emotion` module sends directly messages to the `GUI` whilst ignoring all of AIML.

### 2.6.3 Application Architecture

The game uses a client server architecture (see figure 4). The client is written in unity and the server is a Java application running on Wildfly. Communication between the two applications happens trough a web socket. A web socket is used because it allows the chatbot to be pro-active, which is more difficult with a technology such as REST.

**Source tree** There are two major source trees tracked in separate version control systems. The first manages the client[2] and the second the server[3]. The protocol is tracked separately in the respective client and server folders with the folder name `dto`.

[2]received on commit `40b55c0da1f556ba2b66ea8322d72008c9df1e72`
[3]received on commit `92f12fc26a7da83554903bfe7c6ed1cc64dd5a53`

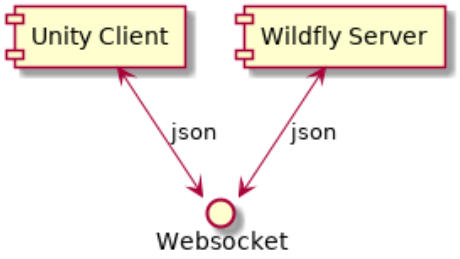


Figure 4: Component diagram of the application

**Protocol** The protocol is setup to be intended for a much larger system. There are hints of a registration system but further inspection revealed that only logging in worked and was required. This is tied into the server's ability to run multiple games. There is also limited monitoring functionality, the active games can be listed with a specialized message. A typical happy path scenario of protocol messages is listed in figure 5.

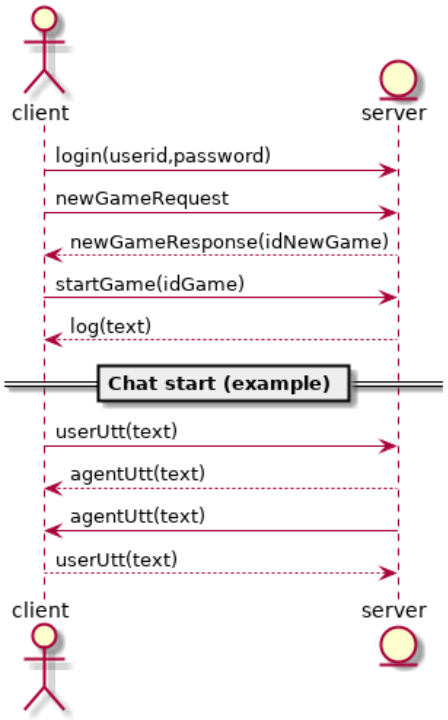


Figure 5: Sequence diagram of a typical game

### 2.6.4 Server architecture

We will discuss the server architecture in more detail since it contains the 'brains' of the application. The most important classes are shown in figure 6. `WebSock-`

`etService` is the entry point for the program where the messages from the client enter.

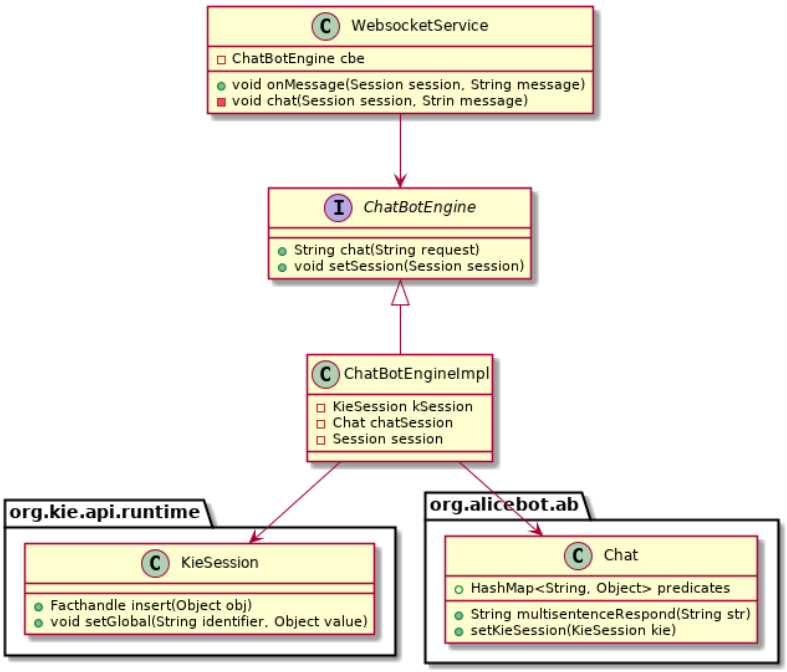


Figure 6: Class diagram of the server, where KIE is the engine that handles the Drools

The `WebsocketService` uses a `ChatbotEngine` to determine how to reply to user utterances, Where `ChatbotEngineImpl` is the concrete implementation. `ChatbotEngineImpl` uses a `KieSession` for the Drools and a `Chat` which is the Alice bot interface. Once a `startGame` message is received the KIE service is started, which runs on a dedicated thread to do drool deliberation. At this point facts can be inserted for the Drools to react upon, in case of the anamnesi scenario the `GameStart` fact is inserted, which is a marker object to indicate that the game has started. This allow Drools to take the initiative, for example when the user hasn't replied after 20 seconds the agent will ask the user why he hasn't replied yet. A detailed overview of construction can be seen in figure 7.

In the class diagram (figure 6), we can see an attribute to the `Chat` class called predicates. This is a bag of variables the Drools can use to keep track of the scenario progression. The `setGlobal` method of `KieSession` is used to expose global objects to Drools. In this case the `ChatbotEngineImpl` is exposed. Insert can be used to insert facts. The difference between facts and globals is explained in section 2.2.4, the summery is that facts are 'just a value' and globals are used as communication with external libraries (for example the `WebSocket` and `ChatSession`).

**Text processing** Text processing is done with help of the Alice chatbot. This bot does the parsing and validation of AIML, with help of the knowledge encoded in AIML it can specify a response. AIML links a pattern to a template,

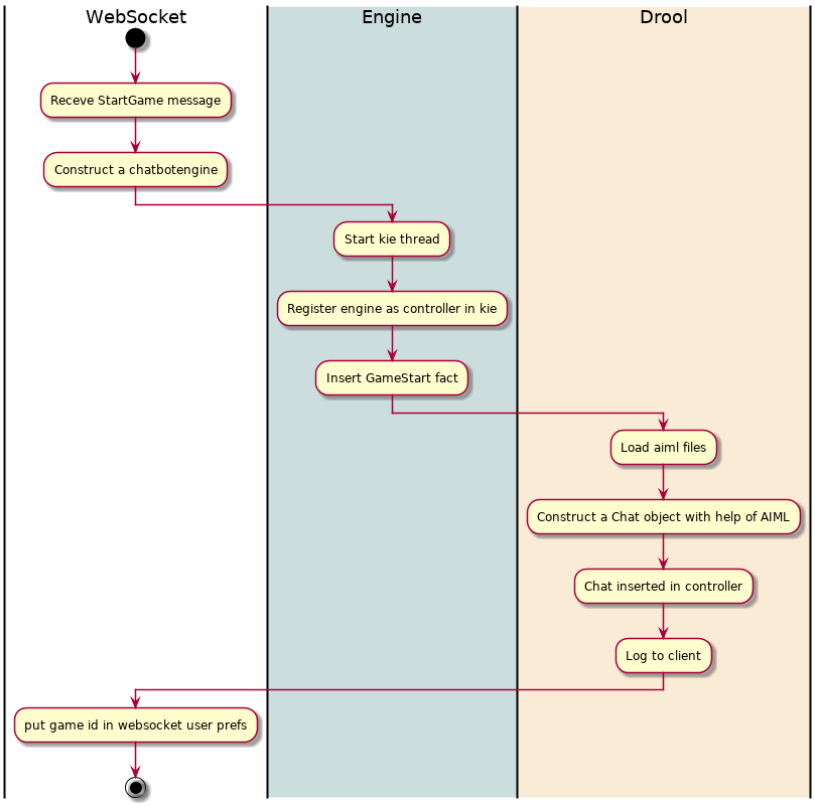


Figure 7: Activity diagram of a server game construction

where the pattern is a user input and a template a response. An example of a pattern template pair can be seen in listing 2.

```
<category>
    <pattern>
        What is the problem
    </pattern>
    <template>
        <srai>why are you here</srai>
    </template>
</category>

<category>
    <pattern>
        * why are you here
        </pattern>
    <template>
        <srai>why are you here</srai>
    </template>
</category>
```

Listing 2: AIML example: why are you here?

In this example the first category indicates that if a user types "What is the problem" (pattern tags), then the answer can be found in a category with pattern "why are you here". The second category does the same but the star indicates that any amount of characters [4] before the pattern can be ignored to match with the category.

**Deliberation** AIML has been extended to allow updating of the Drools knowledge base, as can be seen in listing 3.

```
<category>
    <pattern>why are you here</pattern>
    <preconditions>not healthProblemAsked</preconditions>
    <template>
        <insert packageName="sp.anamnesi.health_problem" typeName="HealthProblemAsked" />
        I'm experiencing a <getDroolsTemplate />. It's quite strong.
    </template>
</category>
```

Listing 3: Extended AIML that uses knowledge

In this case if a user utters the sentence: "why are you here", the bot will check the drool database what his problem is and also update the scenario. Once the scenario is updated the possible responses of the chatbot are changed, as can be seen by the precondition tag. The template tag has some extra tags. The insert tag inserts a fact into the Drools knowledge base, the `getDroolsTemplate` tag queries the Drools knowledge base for a string.

**User utterance processing** An important process to describe is the way currently user messages are processed. Figure 8 gives a detailed overview of utterance processing.

As can be seen in the diagram the message processing happens inside the Alice bot. Tags were added to AIML to allow the drool engine to be updated. The drool system can be relatively easily be bypassed. If there are no tags in AIML the drool system will be oblivious of chat messages. We represented this

[4] It is not really 'any' character, we investigate this further in section 6.5

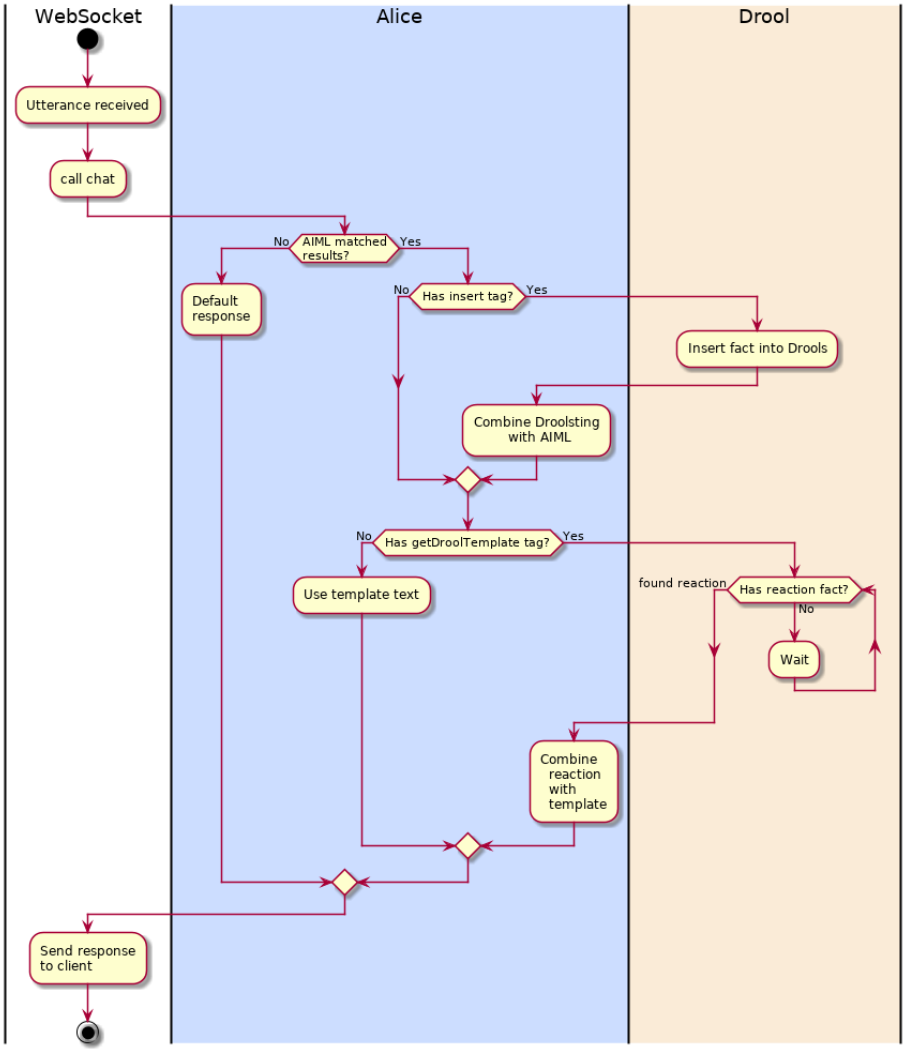


Figure 8: Activity diagram of user utterance processing

situation in figure 9, there is a clear choice between going from a pattern either to Drools or to the template. If there is an insert tag then the 'Drools' state is visited, if not we go directly to the 'Template' state. Then the 'Template' state can use `getDroolTemplates` tags to read information from Drools. Note that there is a loop for the `getDroolTemplates` tag in figure 8. This is because a blocking queue is used, which will block the thread until there is an item in the list. This is represented in the state diagram as the `ReadDroolTag` state.

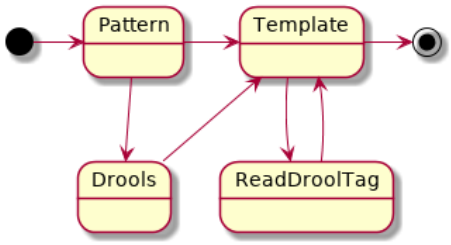


Figure 9: State diagram of utterance processing

# 3 Related work

In this chapter we will give a brief overview of various interesting papers we found during researching this topic. We will start with chatbots in general, each of which has a wildly different approach to perform the same task. Then we will discuss chatbots that also have personality. Finally we will discuss the reasoning behind the direction we chose.

## 3.1 Chatbots

One of the first chatbots was ELIZA, made as early as 1966 [97]. It recognized keywords and based on the linguistic context chose the appropriate transformation. The keyword file and its associated transformation rules were called the 'script'.

The Alice chatbot is a more recent incarnation of the idea, but uses AIML as a basis for the 'script' [91]. Because Alice is licensed under an open source license, and the AIML has been standardized, a legion of other implementations have been made that all can parse AIML. In fact the 'Salve' game discussed in section 2.6, used AIML to deal with natural language. In section 6.1 we discuss our reasons to move away from this almost traditional chatbot paradigm, but in short: We can't use AIML for adding personality unless we'd modify it in such a way that it no longer is AIML. The primary reasons is that AIML selects a response when a pattern is matched, the template, however we want to have the ability to choose between various responses. This 'choice' will then be the personality as a process. We call this the problem of not having deliberation 'space'.

In [98] a sub symbolic chatbot is presented that uses machine learning. It appears that it can handle the general cases of conversation, even questions it didn't train upon. The authors state that its answers are sometimes on the short side, and that slight variance in semantics can result in inconsistent answers. Another issue we have with such a methodology is that it's a completely opaque process. Although you could probably emulate personality by training on specific sets of data, the problem than becomes how would you decide what data is part of which personality? An interesting idea is however to try and use the technique discussed in this paper as a drop in replacement for pattern matching, this is discussed more in section 8.2.5.

### 3.1.1 Multilogue

In another interesting paper [99], multilogue is already possible. However this design is *drastically* different from the one we presented. In this paper they tried to improve input understanding, because it was difficult to hand write in their system, therefore the process was automated. Although in the end there were still several open problems left, such as not being able to deal with what we call templates, or what AIML calls star tags. They also seemed to have problem with context, which we partly solved with scenes. On the other hand it is more advanced in that it can construct sentences, whereas we predefined them.

## 3.2 Personality in chatbots

To simulate personality in communication games there have been already several works proposed. Etheredge used the OCEAN personality theory to create argumentative agents [23]. Although argumentation is not the same as communication, we can consider the method used to make the personality. In this paper a personality model is introduced based on OCEAN. They move from personality values in OCEAN towards action selection with fuzzy logic. Fuzzy logic 'obfuscates' 'crisp' values with more 'natural' terms: Rather than writing $1, 2, 3$, we can use 'low, medium, and high'. With these terms, business rules can be specified. A big advantage of this is that we can modify the definition of the natural terms without modifying the rules they specified. An example of one of these rules is: "if actions is high or self consciousness is high then acceptance is favored".

This has a major disadvantage in that a lot of rules need to be added to do action selection (there are 54 described in the paper), some of which become quite big (for example, some have 7 conditions). This can make action selection opaque. It is for example not immediately clear how a higher anxiety will influence action selection. Having a lot of rules also makes maintenance hard, if for example there is an unwanted behavior many rules need to be inspected before the change can be made.

There are two strategies that could help dealing with this: Modularization which is partly already done in the paper by splitting up action selection and action revision for example. Another approach is to simplify the model, which could be done by using OCEAN traits rather than the facets, reducing the amount of variables from 16 to 5. In fact this will make the argumentative bot a lot more consistent with the OCEAN model. Since some facets were used for action revision and others for action selection, we can have an agent that will revise as if it is very high neurotic but select as if it were very low neurotic. A reason for the complex model maybe the inherent lack of theory OCEAN provides.

Van den Bosch also chose to use OCEAN to model characters in a serious communication game [100]. He used a nested probabilistic if else structure to decide on how agents should interact. His methodologies had some shortcomings however, for example: A not agreeable person was defined as someone who'd had a high probability of telling facts about himself, which in certain situations could be considered strange, for example a spy who was captured. This kind of methodology is called content orientated [1]. Depending on context the personality should change, with which social practices can help.

In [92] an architecture is presented to add personality trough AIML. This paper is interesting because it uses AIML where we explicitly wanted to avoid it (see section 6). The paper does not base itself on a particular personality theory but offers a 'modular architecture' so that the developer can customize them to any particular personality model. This is explained with some examples in [101], it can specify its state and personality in AIML, and then check upon that with 'if, then' rules in the templates. Aside from the fact that this is practically unmaintainable verbose and stretching AIML and XML to its limits (see section 6.2.2), it's also not very modular since now everything is in AIML. Constructs for dealing with modularity such as type safety and even object orientation are just not available if everything is put in AIML.

## 3.3 Campos

Campos used the MBTI to create BDI based agents [1]. In section 2.2.5 we already discussed Campos his architecture. We will use a more fine grained version of MBTI, but his architecture is used, in which personality will be processed orientated rather than content orientated. It is more fine grained in that we use Jungian functions instead of MBTI type labels. The details can be found in section 4.

# 4 Dialogue as a personality process

This chapter tries to answer the question: What is personality from a computationally perspective? We imagine personality being a preference towards a process rather than a preference towards content. We will however not consider yet how to place this in the existing system, but will consider how to model Jungian psychology with BDI into a dialogue process. We want to make personality as a process work, while trying to introduce as few assumptions as possible, and we want these assumptions to be as small as possible. We want to make the system work, while keeping it simple, because simplicity matters [102].

We do this first by analyzing what we want to do in section 4.1, then we propose a rough solution in section 4.2. However since that solution is very rough we use type signatures in section 4.3 to be more precise. This leads us to discuss the dialogue tree 4.3.2 and symbol graph 4.3.3, which are two core components. Then we make a model that can combine individual functions in section 4.3.4. After which we will look into the specific function attitudes and how to implement these as behavior in section 4.4. In section 4.5 we will discuss some changes that were the result of testing. Finally we will consider how this presented method relates to Jungian theory in section 4.6.

## 4.1 Differences from Campos

Campos [1] first considered how to combine MBTI with BDI. His reasoning domain was however in action space (rather than just dialogue), but we still want to use the idea that personality is a preference for a process rather than a preference for content as discussed in section 2.2.5. However rather than using MBTI dimensions we want to use Jungian functions. This is because Jungian function attitudes are the underlying construct of MBTI and several other instruments (such as the PPSDQ and SL-TDI).

There are some differences from the theory discussed in 2.1.2 and Campos' process. The difference is that in the discussed theory we would translate MBTI to the underlying Jungian functions, whereas Campos used the measured dimensions. Translating to the functions has some advantages, by doing so we are for example not bound to just the MBTI. We also get more accurate descriptions of what Jungian functions are, Jung described in his work people with that particular function as dominant. This is harder to do with the dimensions, because if you take an INTJ type and an INTP type the semantics of both the $N$ and $T$ change because of the $P/J$ dimension, as can be seen in their respective order (see section 2.1.2). Campos avoids this by ignoring the $I/E$ and $J/P$ dimensions, resulting in a simplified theory. However we would like to note that it is not an easily extendable simplification. Therefore we chose to translate types to orders in Jungian function attitudes, something which is already done by MBTI.

Another consideration to make is what are these function attitudes? By which I mean what do they represent in computer science terms: programs, objects or functions? What should they be? Since Jung wasn't much of a mathematician [103] it's just an informal definition. However we can make a mapping to certain BDI processes based upon their description, but before that is done we need to make several structural observations. Firstly functions attitudes are not independent, by which we mean that the function attitude

resulting behavior of $a$, followed by $b$ is different than $b$ followed by $a$ (see section 2.1.2). Jungian functions do not have the commutative property. Secondly all functions should be used and their order matters. The first function used should be most prevalent. This means that we can't just execute all functions and do a preference selection on the result.

We will interpret the Jungian functions attitudes as a mapping from an agents believes and senses towards an agent action and new believes. This is then reduced to the scope of a chatbot in the social practice. After this we will look what extra information the function attitudes need in an attempt to reduce the amount of possible believes.

## 4.2 Core idea

Before diving into the type signature approach, we want to give an overview of the core idea. Firstly we see the Jungian functions as a unit of processing. This is a clear design choice, there are alternatives. One could for example choose to make a unit of processing for every possible combination of Jungian functions attitudes which would result in eight factorial different units of processing, or specifically just for MBTI which would result in 16.

We also chose to model function attitudes, rather than functions and attitudes. The reason for taking them as a combination is that there are more precise descriptions available for function attitudes, rather than functions and attitudes separated

A Jungian function attitude as a unit of processing is something where information goes in, the function does its processing and then information comes out. This is analogous to a mathematical pure function. Another way of describing such a process is a transformation upon information. From this we used the idea which MBTI uses too, that these small processing units are in an order, this order determines the eventual personality. What we do is to combine the units of processing into a chain. This chain will then receive the information, which each unit can transform. The information will pass trough the entire chain, to give each unit a chance. The result is then a piece of information too: One part being the reply, and the other being the agents' believes.

There are several phases of processing going on in the entire chatbot. Firstly we have user message parsing, where we try to figure out what the user said. Then, secondly there is action generation, where we use the parsed message to determine sensible replies. After that there is action selection, of which the best action is chosen. This action is finally handled by the surrounding system. The opportunity for personality exists in practically all phases. In the first phase for example we can do filtering based on the type of messages received: A Thinking based personality may filter the message "how are you" as an inquiry based on "how is your disease?", or "why are you here?", whereas a feeling based personality may retrieve a different meaning, as in "how are you doing in live generally"? We chose to not do such kind of personality based filtering because it requires actual understanding of the message received.

There exist techniques such as convolution kernels [104] to decide what was said which can be combined with owl [105] to simulate a sense of understanding. However implementing such techniques is considerably out of scope of this thesis, and even with the existence of such techniques separately, it's still questionable if one can combine them successfully.

## 4.3 A type signature approach

To give a better understanding of the scope of this project we will try to come up with a type signature of a pure function that models all the function attitudes. We do this with a Haskell like syntax [106], in which the arrows indicate a function, left of the arrow is called a domain and the right side a codomain. The domain is also called the argument of a function. If we see a pattern like $a \to b \to c$ means $a \to (b \to c)$ or give an $a$ and return a function $b \to c$, this process is called partial application [107]. Capital letters indicate sets. Note that we have an overview of the symbols used in appendix D. Also note that if we have a function $a \to a$ it does not mean that the *value* of $a$ stays the same, it just means that the type $a$ is used which may change in value. For example think of a function from an integer to an integer that increases the value by four, which also has a signature like that.

We will go from an as broad as possible system (while using BDI) to a precise as possible definition, while still being able to satisfy the domain. This is desirable because it will restrict the amount of computation branches that can happen inside the function. For example a pure function with type $b \to i$ where $b$ is a boolean and $i$ an integer, can only produce two possible integer values, because there is no more input information to make decisions upon. Therefore making the domain as small as possible will result in a less complex system.

To start we'll postpone modeling interplay between the $f_a$ function attitudes and define a type signature for them working individually. To do this we will define some terms, with which we will go from the broadest definition possible towards one that fits the project scope precisely.

Let $\mathcal{B}$ denote the set of all possible believes and let $B$ with $B \subseteq \mathcal{B}$ denote the believes. $\Pi$ is the set of all possible sense information, in which $\pi$ with $\pi \subseteq \Pi$ denotes the perception information. $\mathcal{D}$ denotes the set of all possible actions, and $\Delta$ indicates the set of actions executed where $\Delta \subseteq \mathcal{D}$. With this definition we can define every possible agent configuration[5] as the following pure function type signature:

$$B \to \pi \xrightarrow{f_a} (B, \Delta)$$

This says, we first put in the current believe base, then the sensory information after which we get a new believe base and a set of actions. In this the intentions are encoded in the function used, and the desires are part of the believe base. We marked the $f_a$ arrow, which indicates the deliberation process of the agent, so $f_a$ can be any of the function attitudes discussed in section 2.1.2.

### 4.3.1 Narrowing the model

This definition is however too general for our domain. First of all the set of sensory information can be reduced to a string, since this is the information we get from a user. We can go even further by saying all chatbots do the same thing namely a mapping $\sigma \to \sigma$ where $\sigma$ is a string, where the domain is a user

[5]Note that this is just the deliberation part, there is no memory in a pure function, but the agent's memories can be stored in the believes. The believes can be reused in the next call, it's up to the caller to decide how this happens. This can be done on the thread of control the agent owns for example. Where it will block until a new perception $\pi$ comes in from the environment.

string and the codomain the bots' reply. Therefore we could express all chatbots in the following manner:

$$B \to \sigma \to (B, \sigma)$$

Where $B$ are the set of believes of the chatbot, or its state. We can model all chatbots in this manner because if they don't have state $B = \{\}$.

However a string is still to broad since going from a textual representation to a deliberation process is difficult. Therefore we will introduce another mapping function $g$:

$$\sigma \xrightarrow{g} s$$

Where $\sigma$ is a string and $s$ a symbol where $s \in \mathcal{S}$ in which $\mathcal{S}$ stands for the set of all encoded symbols[6]

A symbol $s$, where $s = (\{\sigma\}, \sigma)$ has the first value as a set of potential returning strings to utter, and the second is the name of the scene the symbol occurs in. The scene name is used as a name space and a crude way to measure scenario progression.

With this we can define another function $g'$:

$$s \xrightarrow{g'} \sigma$$

This allows symbol $s$ to be decoded into string $\sigma$. Note that in this relation there can be multiple $\sigma$ that map to the same symbol, but one symbol produces only a defined set of strings $\{\sigma\}$, that in turn map to itself, on this a random selection can be made.

The simplification is now as follows, firstly we note that $\mathcal{S} \subset \Pi$, since understanding symbols is a form of sensation. Then we can define $S \subseteq \mathcal{S}$ which stands for the symbols the agent understood. This allows the agent to do deliberation without having received any symbol (empty set). Which leaves us with the following type signature:

$$B \to S \xrightarrow{f_a} (B, \Delta)$$

### 4.3.2 Dialogue tree

We have some believes and symbols going in, some deliberation going on and a new set of believes and actions going out. However this type signature isn't enough. To allow the agent to select action in a rational manner, we use a dialogue tree to model the options. The root of the tree is the utterance we deliberate upon. The ply under that is the utterances we consider in response to that. With plies under that in turn being responses to those, etc.

We need to mark which agent uttered what in the dialogue tree nodes, therefore we introduce $\Lambda$ as the set of all active actors, where $a \in \Lambda$. With an actor $a$ and a symbol $s$ we can start thinking about modeling an utterance around which we can model dialogue tree nodes. However to do this, it's important to remember that an utterance always comes with a perlocutionary value set as discussed in section 2.4. Therefore we introduce the set of all encoded perlocutionary speech acts as $\mathcal{P}$ of which a set of speech acts is $P \subseteq \mathcal{P}$. With this we

[6]Originally this was called meaning with an $m$, but we want to avoid confusion with meaning in the social practice, and therefore renamed it to symbol, as in symbolic representation

can define utterance $u$ as a tuple:

$$u = (P, a, s)$$

Where $P$ is the set of perlocutionary values uttered, $a$ is the actor that uttered and $s$ the symbol that was uttered.

Now we introduce $D$ a dialogue tree tuple:

$$D = (u, [D])$$

Where $u$ is the utterance, and $[D]$ is the ordered list of dialogue children. The initial dialogue is just a symbol with an empty list of children. To consider a reply, we would use the same dialogue tree, except with a list of children that is bigger than zero. The most preferred reply is the first element in the list of children. How the actor decides will be discussed in section 4.3.3. An example of an expended dialogue tree can be seen in figure 10.

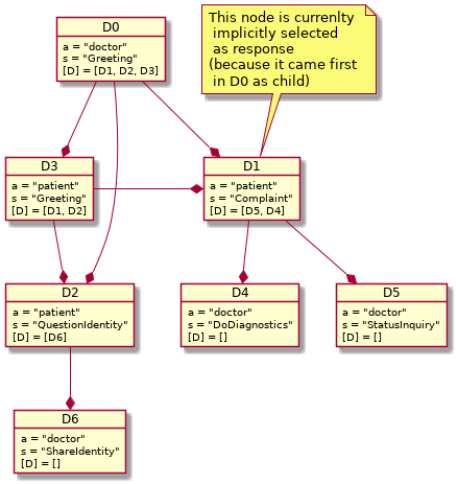


Figure 10: Object diagram of a dialogue tree, at the leaves deliberation stopped.

With this in place we can replace both the $S$ and $\Delta$ with the $D$ and $D$ respectively, we can also remove $t$, since it's now contained in the utterance. This is convenient because now we can model function attitudes as processing units that take a dialogue tree and modify it. We are left with the following type signature:

$$B \to D \xrightarrow{f_a} (B, D)$$

So we receive a dialogue tree from the user, which can just be a root node, and then after processing we put out a dialogue tree plus the replies which are the children, whereof the first child is the most preferred. Note that this $f_a$ function is an endomorphism, meaning that the input arguments are of the same type as the output arguments. We annotated the output arguments with $t + 1$ to indicate they could've been changed, not to indicate a different type.

Now we should note that this type signature heavily constrains our agent. It for example can't handle being punched in the face by the doctor unless there is a symbol encoded for that. It also runs into trouble when the agent is asked

to sit on the counter. Movement should be possible. However once movement becomes a requirement we can just create a new function and type signature that is less restrictive. This new function can still use these functions we are modelling now for dialogue.

### 4.3.3 Symbol graph

To make sure the agent stays on topic we will make use of a symbol graph. This graphs gives connections to the symbols described in section 4.3. The meaning graph $G$ is a set of connections $c \in G$ where $c = (P, A, s_1, s_2)$, $s_1, s_2 \in \mathcal{S}$, $A \subseteq \Lambda$ is the set of agents that can use the connection, to prevent cases where the patient asks the doctor about his health problems. $P$ is the perlocutionary value set of the speech act, as introduced in section 4.3.2. This is encoded in the edges because it's not the meaning that causes these but the way you get to those meanings. In other words, being polite and then telling bad news causes different perlocutionary values than just telling bad news.

From this we can define a function that gets the allowed connections using a symbol and an agent from the graph:

$$G \to a \to s \to \{c\}$$

We can retrieve $a$ and $s$ from the current node we are processing in $D$. The result is a list of connections we can go to from that symbol. We can map a connection $c$ to a utterance $u$ by flattening the set of Actors $A$ in connection $c$ into individual actors, for each actor we can create a possible utterance $u$ from the information in $c$. From these utterances in turn we can create new dialogue tree options.

The introduction of the symbol graph is probably the most radical change this thesis proposes. It moves chatbots away from the idea that responses are many to one relations always and opens up many to many relations. There are more advanced techniques such as owl available, we discuss these in section 8.2.4. We didn't use that because we thought the step from ontology to language would be to difficult to finish in time. The symbol graph provides a good middle ground, in which it's relatively easy to implement but offers enough freedom to encode personality in as a process. Note that this approach fits into the information state transitions discussed in section 2.5.

### 4.3.4 Function attitudes combined

The first thing a programmer may think of when trying to combine behavior is functional composition. The most important requirement for this to work is that the input type and output type need to be the same of the two functions we want to combine. What is problematic however is that using functional composition in this way would make it impossible for function attitudes to inspect results of their auxiliary functions. This is an important feature we want to keep because if for example a judgement function is first in the order of functions and receives the user meaning it can't do its job yet, more on his in this section 4.4. Therefore we consider another approach.

We considered storing the functions in a list and then let an external control unit decide which function processes next. However this would leave the control of the function being called outside of the control of the function attitudes,

therefore personality wouldn't play a role in deciding the function being called. It will also create another problem of deciding when a function is called. So to solve these problems we looked at another possibility.

In this approach we will give $f_a$ another argument which is the next $f_a$. This looks like the following:

$$\left(B \to D \overset{next}{\to} (B, D)\right) \to B \to D \overset{f_a}{\to} (B, D)$$

Note that the function in the next bracket has the same prototype as the codomain. A more compact way of representing this type signature is the following:

$$\overset{next}{f_a} \to f_a$$

In this case the *next* function can play an advisory role to the codomain. A unit function can be defined that produces empty sets as results for both believes and action. By unit function we mean the initial *next* function that does nothing and just returns the believes and dialogue tree.

To illustrate the use of this type signature design more clearly we'll sketch an example with the first two function attitudes of the INTJ type:

$$N_i > T_e$$

So to encode this as a function we start with the least preferred function attitude namely the $T_e$, however to let it play an advisory role in the $N_i$ function we first need to complete the *next* argument. Because it's the least preferred function we just use the unit. Now the partially applied type of $T_e$ satisfies that of $N_i$ and we can use it as *next*. This methodology can be used for an entire personality type (all eight functions in some order). Also as an analogy we could say that we're dealing with an intrusive linked list. The next argument is the next item in the list. Unit is the tail item of that list, which exists to provide a start point to create the data structure upon.

With this methodology function attitudes can decide themselves to consult the next type. Then they can inspect the result, and even the changed believe base to decide if it's a good idea to use the result.

This architecture can be extended with the scale based Jungian models such as SL-TDI and PPSDQ by introducing a random choice for using the current or next function. However this becomes rather messy because we're modeling pure functions, therefore we leave this as an exercise to the reader.

## 4.4 Applied to Jung

Up until now we modeled the type signature to have a dialogue tree as input and output. However we have not considered how children are generated and how the order is determined. If we look at the definition (section 2.1.2) of rational and irrational, we can make a design decision about what these functions should do to the children. Rational functions are about making decisions therefore they should apply order to the children. irrational are about producing information therefore they should produce new children.

There are however some edge cases to consider when modeling this idea. Say the primary function is a rational one. It receives a dialogue with just the

root node. Currently it cannot apply any order since the children list is empty. Luckily it can still use its next function, which is irrational (see section 2.1.2).

Another situation to consider is what to do when there are already children. Should an irrational function extend this list of children or go to some leaf node? Same question for a rational functions should it sort everything or just the children list on its respective level. At which level a function should operate is rather fundamental. We will discuss this level of operation in more detail at section 4.5.2, since this discussion is quite complicated and not important for the main idea of what rational and irrational ought to do.

With this in mind we can still say these things about the conceptualized architecture: *rational* functions change the order of possible replies. *irrational* increase the number of children. So if we start with an irrational function it produces several related symbols to the inputted dialogue tree. The original symbol uttered by the user is the root node and the produced response symbols are the children. These then get inserted into the next rational function which modifies the order of the children. This continues until all functions in the personality had their chance. Finally the unit function just returns the Dialogue and believes without modifying them, which returns trough all functions from before that can still modify the result. This could happen if a rational function was the first function for example and didn't have any choices available to decide upon.

### 4.4.1 Irrational as a process

The irrational functions rely heavily on the symbol graph to create new children in the dialogue tree. This is under the assumption that connections in the symbol graph are always on topic.

In the initial design of the $S$ and $N$ functions, we considered them in the following way: $S$ would be analyzing all available options rigorously in a forward chaining process, whereas $N$ would do backward chaining, starting at the goal and going trough some way points directly to the starting point.

This would translate into $S$ going several plies deep into the symbol graph before calling the *next* function and returning the result, and if we assume that the *next* function brings us closer to the goal we can use it as a heuristic to let it determine the direction for $N$. This of course doesn't allow us to do backward chaining since there is hardly a guarantee that the *next* function will bring us back to the origin, in fact we may get stuck in a loop.

Alternatives to the implementation proposed include the use of probabilities to determine appropriate responses. However this introduces a new problem of how to obtain the probability distributions. Machine learning could be used for this, but this raises the question: 'Learn on what?' Since the answer to that question is non-trivial, we consider such a solution out of scope.

We decided to use a more simple approach instead, $S$ would be options in breath, analyzing many details around it, and $N$ would be options in depth, just taking what first comes to mind and plan ahead on that.

**Intuition** We can consider $N_i$ to be a depth first approach. Going several plies deep and at each ply consulting the *next* function which step to take. $N_e$ on the other hand just takes the top $x$ of the current dialogue options and expands those, but then next step it will again consider the entire existing tree to find

the best $x$ of each ply. This will of course be a much more shallow consideration than $N_i$, but also more broad. Which is the behavior we are looking for in both $N_i$ and $N_e$ (see section 2.1.2).

**Sensing** The $S_e$ function just receives all possible connections from the current meaning for several plies and then applies the *next* function on it. The $S_i$ however is more conservative and will only pop $x$ random meanings by default (the first $x$ connections), however it will construct its own connections of whatever the user said in response to the bot from previous conversations when at the same meaning (if it didn't exists already). Whenever these connections are available they will substitute the random $x$. $S_i$ starts of kind off similar to $S_e$ but builds up over time. So $S_i$ acts as a learning function and $S_e$ as a possibility function which is what was described in the theory of section 2.1.2.

### 4.4.2 Rational as a process

In the current design the rational functions apply order to the children of a current dialogue node. Then once finished they will call the *next* function on the most preferred child. This is to ensure all function attitudes can do some processing.

Please do note that although we have a game tree, we're not dealing with a zero sum game. Dialogue is cooperative rather than competitive (see section 2.4). So doing an algorithm such as mini-max is out of the question. However we will borrow parts of it. Namely whenever a rational function finishing ordering the input set it will call the *next* function to do deliberation on the most preferred item.

We also model the rational functions as local optimizing functions. Only the current ply and maybe the next ply is considered, but not the entire tree. The primary reason for this is time constraints. However there is no reason why the entire available tree couldn't be used.

**Feeling** Initially we wanted to create two lookup tables for both feeling functions one. However this would be confusing to configure, the scenario creator would need to decide which values are external and which are internal. Campos however modeled feeling as a prediction of what the other agents will do. This describes $F_e$ rather well, $F_i$ not so much however. So we adapted and adopted that idea for $F_e$ and for $F_i$ we used the lookup table.

Both feeling functions $F$ use the perlocutionary acts to order the children. $F_i$ uses a predefined value set $h$:

$$p \xrightarrow{h} i$$

. Where $p \in \mathcal{P}$ is a perlocutionary value. This valuation is done by a lookup table on all available perlocutionary speech acts. $F_e$ tries to figure out what the conversation partners values by picking the perlocutionary act the other chose most. This is done by simply keeping track on how many of such speech acts the partner uttered and picking the that has been uttered most, if that one is not available we move to the next one. This is similar to fictitious play [108].

**Thinking** Normally the $T$ function is about reasoning. There is little reasoning to do in our scenario except to get to the goal as soon as possible. The thinking functions $T$ do this without paying any attention to perlocutionary speech acts.

We could say that while feeling is concerned with perlocutionary speech act goals thinking on the other is concerned with symbolic goals. To model the goals of the thinking functions we will introduce the set of goals in an agents believe base $\Phi$. Where a single goals $\phi \in \Phi$ consists of $\phi = (a, s)$ a symbol uttered by a particular agent. Then there also exists the function that can compare goals with each other:

$$\phi_1 \to \phi_2 \to b$$

where $b \in \{\top, \bot\}$ is a boolean, true or false that determines if the first goal is more important then the second. This function is asymmetric. Finally there is a function that determines if a goal is completed or not:

$$\phi \to b$$

Now to begin with $T_e$. It sees the conversation as the problem to solve. Therefore it will consistently choose speech acts that could help the partner to progress the scenario. It wants to put the partner in a position where he has almost no other options except to progress the scenario. If it encounters a child node with a goal $\phi$ in it it will give priority to that. If there are multiple goals in the options the comparison function can be used to determine the most important one. Scenario progression is measured with help of scenes. If an option changes scene we assume it progresses scenario. This comes secondary to finding goals.

To model $T_i$ however the most obvious solution would be to implement an axiomatic logic system. This is rather heavy on maintenance. Every agent would need to have their own axiomatic system to determine what to do for each node in the symbol graph. The only real solution would be to create this dynamically somehow, but this is out of scope of this thesis. Therefore we looked for an alternative.

$T_i$ wants to help the conversation partner to analyze the problem according to the partner's own internal logic framework, and to do this it wants to give as much options as possible to the partner. Therefore it will choose the speech acts that produce the most symbols for the partner. To do this it will sort the child nodes according to as much unique symbols as possible. Options that are goals still get precedence however.

### 4.4.3 Believes

Now you may argue at this point we haven't refined our types a lot, since the believe structure was defined as 'Every possible believe', which is basically analogous to 'Anything you can think of' or in a object orientated terminology: `Object`. Since the believes serve as input of our function and output of the function we may as well have said $Object \to Object$. Of course the believes are not intended to be direct program output but rather just part of the mind. In other words, the believes are intended to be kept in a container whereas the input $D$ and the output $D$ would only be visible for the 'outside world'. Still we want to refine believe to something which is less broad in scope. To do this

we analyzed the Jungian functions and see what 'extra' information is required to function to perform their operations.

We listed the function attitudes $f_a$ and their required information into table 2. Therefore $B = (h, [u], \Phi, G, a, G', h')$. For reference a symbol table of all introduced symbols is shown in table 9 in appendix D.

| Function | required data |
|---|---|
| $T_e$ | The set of goals $\Phi$, scene information and $G$ |
| $T_i$ | The set of goals $\Phi$, and $G$ |
| $F_e$ | Utterance history [u] and $G$, self believe $a$, learned values $h'$ |
| $F_i$ | Personal values $h$ |
| $S_e$ | $G$ |
| $S_i$ | Utterance history $[u]$ and $G$, and learned graph $G'$ |
| $N_e$ | $G$ |
| $N_i$ | $G$ |

Table 2: Function attitudes and their required data.

## 4.5 Practical changes

In this section we discuss what influence testing had on the application. A big change was the way how turn taking operated discussed in section 4.5.1, secondly the way we combined functions in section 4.3.4 has some issues with depth discussed in section 4.5.2.

### 4.5.1 Turn taking

In the naive approach we modeled turn taking with a simple round robin strategy. Basically the irrational functions would only consider options that change actor between plies. This makes it difficult however to model agents that hold long monologues, which happens for example to Susie in the case study (see section 7.1.2). You could do it by making just more symbols that hold all these utterances in one. However this is very inflexible. So to solve this problem we use alternation of actors whenever there is a tie between two options. So irrational would leave out the option that doesn't alternate, and rational would prefer alternation when possible.

### 4.5.2 Function ply depth

A big issue that turned up trough testing is at which level a function ought to operate. We have a two pass architecture, where functions can inspect the dialogue tree before passing it to the *next* function, but they can also inspect the result of the *next* function. The reason for the two pass architecture is explained in section 4.4. Note that at some point in the reference implementation we stepped away from doing a pure *next* based approach and we re-introduced the list mechanism that was described in section 4.3.4. This was to allow drool rules to do inspection of the personality process in between function attitudes, for example to allow emotions to have their influence, or norms from the social practice. Partly because we have a hybrid approach of deciding the *next* function, and because we simply hadn't worked it out for the pure *next* based

approach we need to answer the following question: "How does a function know at which level in the dialogue tree it should operate?"

In a naive approach we tried an implementation where irrational functions will by default go down the left (most preferred) path to a leaf node and then generate more options, whereas the rational functions would sort the one layer above the leaf layer. This has a problem in that it would make a rational function in the first position of a personality the least relevant function, since in the first pass it does nothing and when going back it works at one level above the leaves. This is a problem because it should be the most relevant function instead of least relevant.

Another approach is to use outside information to determine height. Basically we would put into the believes the order of functions. With this information and the dialogue tree we can calculate the right level to operate upon. A question that remains is: Should the rational function sort everything even options below its level or just options in its level? We decided that rational should sort its level and everything below it, because it allows the dominant rational function to have a more pronounced effects, whereas deeper level rational functions don't have a direct effect upon the resulting dialogue tree. The 'deeper' less important rational functions only have a guiding role for irrational functions.

We could also let the rational functions sort the entire tree, and let irrational always extend the most preferred option. At first glance this idea would make order for rational functions irrelevant. Perhaps this isn't the case however, since a lower level rational function would still guide which part of the tree get extended.

To sum up, there are two methods of dealing with this issue. Firstly we can let rational functions sort everything, but then the deeper rational functions will become less relevant. Secondly we can let functions operate at a particular level based upon their position in the personality. We chose to do the latter, because we thought this would make earlier rational functions more influential. With this particular choice we can also make a decision about whether a function should operate at a particular height, or go downward trough the entire tree, we chose to let it go downward because then the personality will be more consistent in its choices if it wants to utter lower level replies. Note however that deeper rationale functions can still have effect by virtue of deciding which actions are generated indirectly.

To calculate an operation height, we need to know the function order, then the function itself and finally the height of the dialogue tree. Which results in the following:

$$[F_a] \rightarrow F_a \rightarrow i_{D_{\text{height}}} \rightarrow i_{\text{operate level}}$$

Where $F_a$ is the Jungian function, and $[F_a]$ is the personality, which consists of an ordered list of Jungian functions, $i_{D_{\text{height}}}$ is an integer which indicates the height of the dialogue tree and $i_{\text{operate level}}$ is the suggested operation height. To do this we group the functions in function attitude pairs, a rational and irrational function combined into an tuple. Of this we take the pair index of the input functions' function pair, plus one if the second value of the pair is rationale, *and* the input function is rationale, otherwise plus zero

## 4.6 Consistency with theory

In this section we will explore if INTJ and ENTJ (MBTI) types would produce different actions by analyzing when the functions would act. We will only look at the first two functions because the argument holds for all functions after these. The first two function attitudes of INTJ are:

$$N_i > T_e$$

And of ENTJ they are:

$$T_e > N_i$$

What we would expect is that the $T_e$ and $N_i$ produce different results because of the order they have in the sequence. If we assume personalities only have these two functions, their respective differences are:

- INTJ: At each ply $N_i$ will use $T_e$ to select the options generated.
- ENTJ: $N_i$ will generate random options at each ply, which $T_e$ sorts recursively.

INTJ and INTP are different in attitudes, but have the same order. Since attitudes produce a different process by definition (see section 4.4), we can conclude that they will also behave differently.

Because we have behavior in dialogue trough order of function attitudes we can consider this system consistent with the theory MBTI presents. It is also consistent with Campos' work because the functions are just units of processing that can be combined, and therefore we have personality trough a process. Thus this system can be considered consistent with the major sources of theory we used.

# 5 Architecture

To combine the ideas discussed in section 4 with the existing program, some big architectural changes were introduced. For example the Alice bot was completely removed in favor of a new less tightly coupled scheme. The Drools engine has become the center of deliberation (which previously was the AIML). We will discuss these changes in this chapter.

In this chapter we will discuss two architectures, the first is the architecture which is actually implemented, this deals with a single agent and the user. In section 5.1, we will describe the main architectural changes between the current implementation and the original architecture discussed in section 2.6. After that we discuss the data structures in section 5.2. In section 5.3 we discuss how the initialization of the program with help of the discussed data structures, we continue discussing the normal operation of the program in section 5.4. In section 5.5 we discuss how social practice support can be added in the future and in section 5.6 we do the same for a multilogue architecture, in which multiple agents can participate in the dialogue.

There are also several items we won't discuss in this chapter because they haven't changed, these include the protocol, and the Wildfly server and the unity client.

## 5.1 Overview

A deployment diagram of the architecture can bee seen in figure 11, where the dashed arrow means constructs, the solid arrow means uses and the other lines mean interacts.

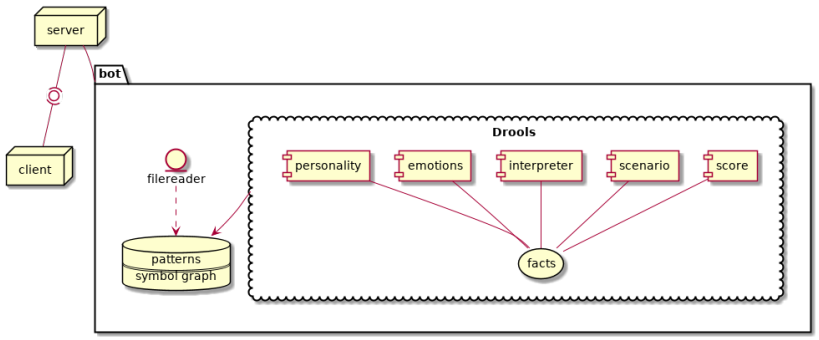


Figure 11: Deployment diagram of implemented architecture

In figure 11 we can see the new deployment diagram. The server and client have mostly stayed the same, except for the bot. This has been completely replaced by a new system. The Alice bot used to be a file reader, pattern database and deliberation engine in one package. These concepts have now been split up, the file reader handles the loading part of the bot, after that it just inserts the symbol, pattern and connection databases into the Drools rule engine. The Drools rule engine then handles all deliberation, which can happen with the various components, such as personality emotions etc. Even the pattern matching is done by a drool rule with help of the pattern database.

Since the file reader is no longer a part of the bot, it should be easy to add support for other data formats.

The biggest difference from the original architecture is the removal of distinction between Drools and the chatbot. In the new architecture we make all information in the files available to the Drools in a database. This is starkly different than the architecture used in section 2.6.4. In the old architecture, the reply for a message is already determined before Drools had a chance to do deliberation. What's even worse is that if the Drools want to utter a spontaneous utterance, then it had to be encoded in a string inside the Drools themselves. This means the strings facing the user are spread over both the AIML files and the Drools. This is confusing for new scenario creators since completely different folders have to be accessed to change the strings.

The changes proposed here, result in a much more simple architecture. Only one place does deliberation rather than two and only one API is used for generating responses, whereas previously Drools could generate replies, *and* the AIML bot.

Note that although we removed the ability for the bot to use AIML, it should be relatively easy to convert from the old AIML structure to the new format with help of a script. A proof of concept of this has been made of this in section 6.6.4.

## 5.2 Data structures

> Bad programmers worry about the code. Good programmers worry about data structures and their relationships.
>
> – Linus Torvalds [109]

In this subsection we will discuss the main data structures used to implement the ideas from section 4. We use class diagrams to accomplish this which are based upon UML [110].

Before this is done we would like to point out several things to keep in mind. Firstly, we do not show everything precisely as implemented in the code, because that would clutter the diagrams. What we do model is all relevant information in structures and the relationship between those, following the words of Torvalds. Secondly, it's better to think of the classes shown here as value types, in other words: The model part in the 'model view controller paradigm' [111]. Note that we use public fields in cases where immutability was possible. Thirdly we split up the class diagram into several to save space, the model has become rather big. However there exists a similar separation like this in the source code, the lower level components are in the `salve_drools` projects, whereas the higher level components are in the `salve_personality` package. The reason for this separation is that currently, the bot will simply not function without the low level components, but it can function without the high level components. Which in practice is done with the low level replies. Finally note that we use $[\dots]$ for lists and $\{\dots\}$ for sets in the class diagrams, to save space.

### 5.2.1 Low level diagram

In figure 12 the diagram containing the low level data structures used. These are the basic assumptions, or building blocks the implementation is constructed

from.

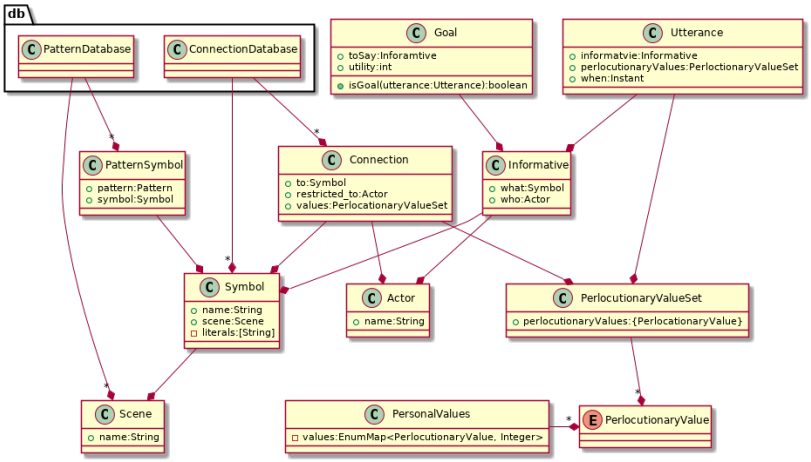


Figure 12: Class diagram of the low level model

From figure 12 we can clearly see the importance of the `Symbol` structure in the application. Simply by counting the amount of structures that consist of it, and of course it is a very important structure because it is the building block that we use for information state transitions. As described in section 4 we map strings into symbols, and once the symbol graph is used to find a connected symbol, we can map symbols back to strings again. An overview of the relationship between the theoretic representation and the implementation for this class diagram can be seen in table 3.

| Symbol | Corresponding Class |
|---|---|
| $\sigma$ | `String` |
| $s$ | `Symbol` |
| $g$ | `PatternSymbol` |
| $g'$ | `Symbol`[7] |
| $P$ | `PerlocutionaryValueSet` |
| $p$ | `PerlocutionaryValue` |
| $a$ | `Actor` |
| $u$ | `Utterance` |
| $c$ | `Connection` |
| $h$ | `PersonalValues` |
| $\phi$ | `Goal` |

Table 3: Overview of section 4 symbols and their class representations

Something that was thought about is how similar a `Connection` is to an `Utterance`. Except for the `instant` field, they are the same (note that the `informative` field of utterance is the same as the `to` and `restricted_to` fields of connection). However their semantics are clearly different: A connection entails

[7]The literal strings are used for back conversion, in combination with the `MatchedQueryDB` described in section 5.2.5

a possibility of an utterance, but it does *not* mean it will be uttered, whereas an utterance is a used connection, that became a realization. Therefore, we consider this type level distinction as correct.

These considerations become especially important when structures are essentially the same, as we can see with the `Scene` and `Actor` classes. The only thing they contain is a string. Even the field names are the same! Are we correct to treat these as distinct types? We argue yes because they entail completely different semantics, the `Scene` class is used to group symbols and patterns, whereas the `Actor` class is used to identify actors.

The next question would be: Should we use an inheritance relation to make our code more DRY (don't repeat yourself) [112]? For by example introducing an abstract class `ANamed` and letting `Actor` and `Scene` be extended from those. We argue no, because it introduces more complexity than that we would save on code reduction. It would also open up the possibility to use the implicit covariant relationship, resulting in functions that could accept an `ANamed` argument for example. As soon as client code starts using that, the single inheritance 'slot' Java provides is occupied forever, or at least until a major refactor occurs. Therefore we didn't do this

### 5.2.2 Believes and dialogue tree

In figure 13 we can see the higher level structures of `Believes` and `DialogueTree`. Note that we significantly simplified all classes from section 5.2.1 in this figure to save space.

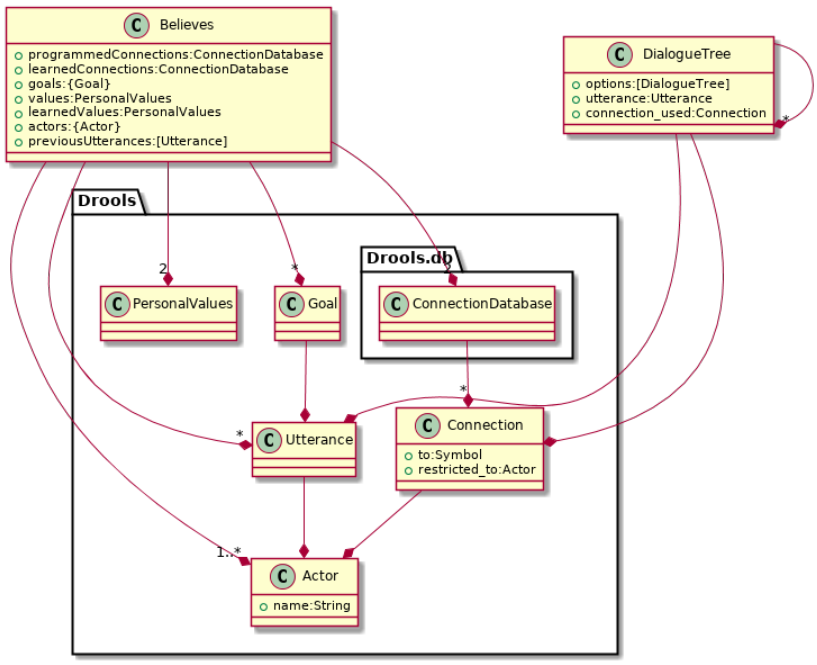


Figure 13: Class diagram of the high level model

From figure 13 we can see the main clients of the low level drool package is

indeed the `Believes` class and after that the `DialogueTree`. `Believes` provide the Jungian functions with bounded information about the mind of the chatbot as discussed in section 4. In table 4 we can see the relationship between theoretic representation and that of this class diagram, excluding the ones discussed in previous section.

| Symbol | Corresponding Class |
|---|---|
| $B$ | `Believes` |
| $D$ | `DialogueTree` |

Table 4: Overview of section 4 symbols and their class representations

The `Believes` structure is very peculiar, because it doesn't represent a single idea or use case. Instead it's just a combination of various elements that are required for the Jungian functions to operate. But none of the functions use *all* fields, so they get more 'assumptions' than they need, which is an architectural problem. The drool fact base and rule 'when' clauses have a mechanism for dealing with this as described in section 2.2.4. So an argument can be made to remove the `Believes` structure and replace it with the drool fact base and 'when' clauses. This hasn't been done, because it would be a very invasive operation, currently the Jungian functions have the `Believes` structure in their signature. This can then be replaced by what they individually need, rather then what they as a whole need. With that change the personality functions could be flattened to drool rules, which would make the architecture even more simple.

The `DialogueTree` structure is however a whole other beast. It provides a well defined structure, and some utility methods that make tree navigation much easier. These methods aren't shown in the figure because their type signatures are rather big.

### 5.2.3 Db package

In figure 14 the databases are shown. This is a sub package of the model. The database is an immutable hash map. It also provides some extra Java 8 features, such as returning an `Optional` rather than a null reference as get method. The concrete implementations of database can add extra behavior once the type value of the generic parameters is known which is done by connection database for example.

`SymbolDatabase` is the first database constructed during the initialization phase. From this the other two databases can be more easily constructed since they can lookup symbols in the `SymbolDatabase`, rather than worrying about construction of new ones. This class is the realization of $\mathcal{S}$ from section 4.

`PatternDatabase` can store patterns per `Scene`. There are two constructed of these, the first one constructs the patterns that are in a scene, and the second one constructs patterns that are of scenes where the current scene is connected to. For example if there exists a connection form a symbol in scene $a$ to a symbol in scene $b$, the patterns from the symbol in scene $b$ are stored in the key of scene $a$. This second database allows scene transitions to occur. To construct this second database a connection database is required however.

`ConnectionDatabase` is a database that stores a connection set from a symbol. This class is analogous to the symbol graph $G$ from section 4, it is used

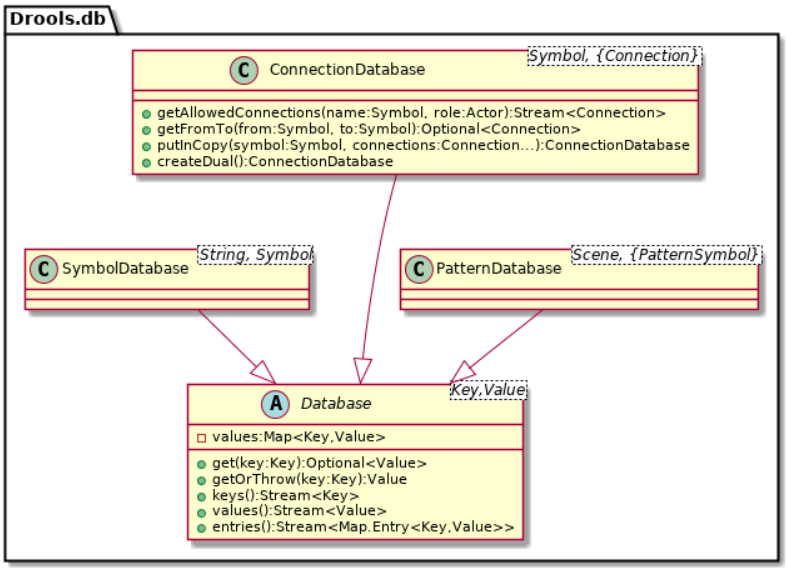


Figure 14: The database package

to determine what the bot could say, and what it thinks its speech partner can say. This database has a special method named `createDual` which is used to create a connection database where all the connections are flipped. This is used to create the second pattern database, making looking up the required patterns much easier.

### 5.2.4 Jung in Java

To implement the theory presented in section 4, several issues had to be overcome. First of all Java has no native support for doing partial application. We worked around this Issue by introducing a structure that contained the arguments of the $f_a$ function described in section 4.4. The structure is called `JungFuncArgs` and can be seen in figure 15. The next issue was doing Functional composition, and although we can do this in Java with anonymous classes, we wanted to make the relation more explicit. The `NextFunction` and its respective field in `JungFuncArgs` is this explicit relation. Adding this field to the `JungFuncArgs` makes the functions a true endomorphism, although it deviates from the theory since the result now also has a next function. This also introduces an infinite creation sequence, there always needs to be a next function. To break this the `UnitNextFunction` was introduced. An argument can be made for using the `null` reference instead, however this is considered a bad practice [113]. Finally for testing purposes we needed to be able to inject other functions than the ones defined in the `JungianFunction` enum. Therefore the `JungFuncAccessor` interface was introduced. This allows unit test to check if the next function was called for example, but the architecture also becomes more extendable because of this.

Figure 15 shows the elements required for Java to apply an $f_a$. To do this we first create a `JungFuncArgs` structure with its `create` method. Then we

insert the Jungian functions we want to apply. This is a list of elements of the `JungianFunction` enum, these elements aren't shown in the figure because they're just the abbreviated names of the Jungian functions, for example $S_e$ for extroverted sensing. The `insertNextFuncs` returns a new `JungFuncArgs` object with the inserted next functions, these next functions are not evaluated. Also note that `JungFuncArgs` is an immutable object, so the result of the `insertNextFuncs` needs to be used. To apply the function we use `applyNext()`, which returns a new `JungFuncArgs` object with the resulting values.

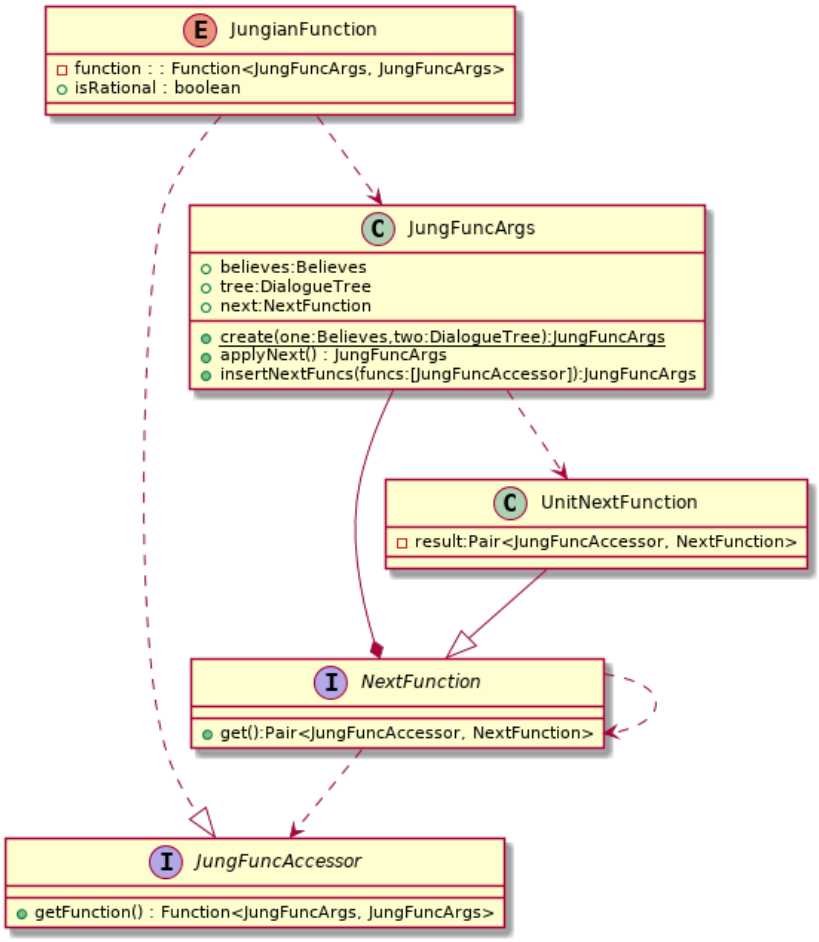


Figure 15: Jung in Java

### 5.2.5 Before and templates

After the personality was implemented, we wanted to bring the bot up too feature parity with the Alice bot. To do this several new data structures had to be introduced which can be seen in figure 16.

We can see from figure 16 that the consumers of these extensions are the `Symbol` class, the `Utterance` class and the `Connection` class. What also can be deduced is that the before extensions was probably a lot easier to realize than

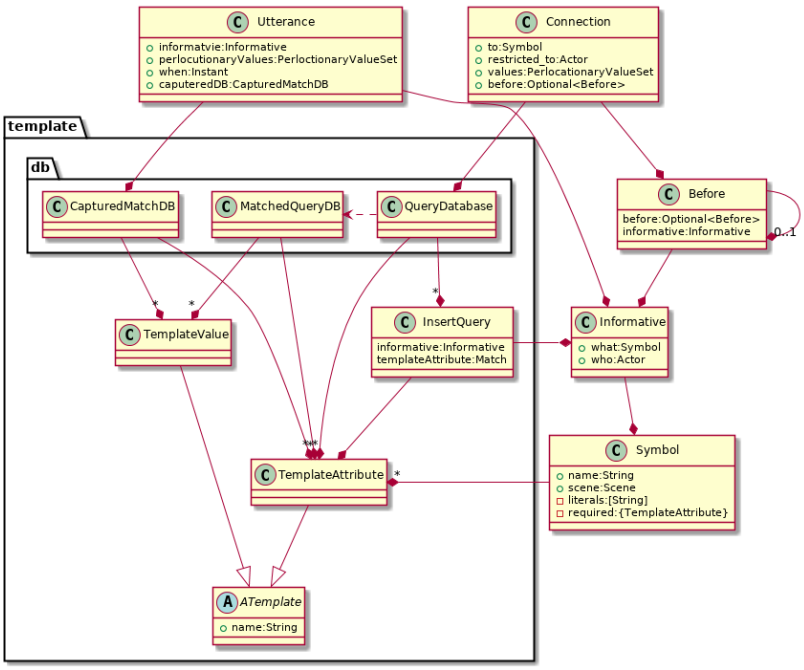


Figure 16: Before and template class diagram

the template extension, simply by counting the amount of classes it introduced and modified. Whereas the template required the modification of at least three existing data data structures, the before only required to modify the `Connection`.

So the before class is self recursive, something which we've seen earlier in the `DialogueTree` class for example, however this is just an optional self recursive relationship. What it does is lay a restriction on `Connection`, the `Informative` in the optional `Before` has to be uttered before this connection can be used. See section 6.4.4 for a more in depth explanation.

The template system does something else. It introduces the ability to match variables from the regex and re-insert these as a template into existing symbols. This is explained in depth in section 6.5.

### 5.2.6 Support types

Because we are working with Drools, we often use a technique of wrapping values into other types, to signify their progress in Drools deliberation. Basically we use types as labels to indicate progress. These types can either be defined in Java or Drools. If they are defined in Java, both Java code and Drools code can use it. If they are defined in Drools, only Drools code can use it. In figure 17 we can see the supporting types defined in Java and the relations they have with types defined in previous sections. We can see the types that are defined in the dialogue Drools package in figure 18 and that of the personality specific Drools in figure 19.

The classes described in figure 17 have the primary function of starting the

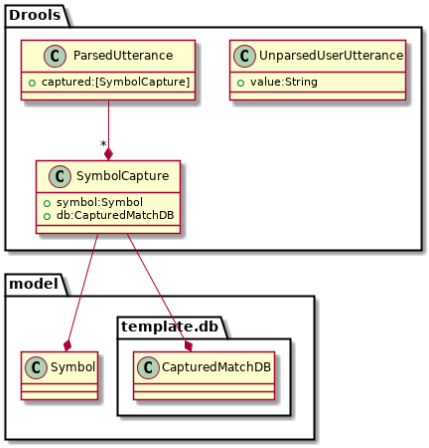


Figure 17: Supporting types in Java

deliberation process. With `UnparsedUserUtterance` the initial utterance is inserted, and with `ParsedUtterance` it is translated to an understood symbol list. With these symbols the `CaputerMatchDB` is stored, which is later used to create an `Utterance` from. This isn't done immediately because the `Believes` structure is required to create the utterance. We for example need to know which connection was used to get to this point in the conversation to figure out the `PerlocationaryValueSet`.

By studying figure 18 we can start to understand what is going on inside the Drools. We can for example see that a distinction is made for when a result matches an in scene pattern or a neighbouring scene pattern. These structures are of course there to do scene switching. We can also see that to create a reply we need to have a `QueryDatabase`. This is the result of the template match searched in the utterance history.

In figure 19 we can see the drool defined `PersonalityProcess`. This structure tracks traversing the Jungian Functions. We manage this inside Drools to give other rules the opportunity to inspect the deliberation process while it's going.

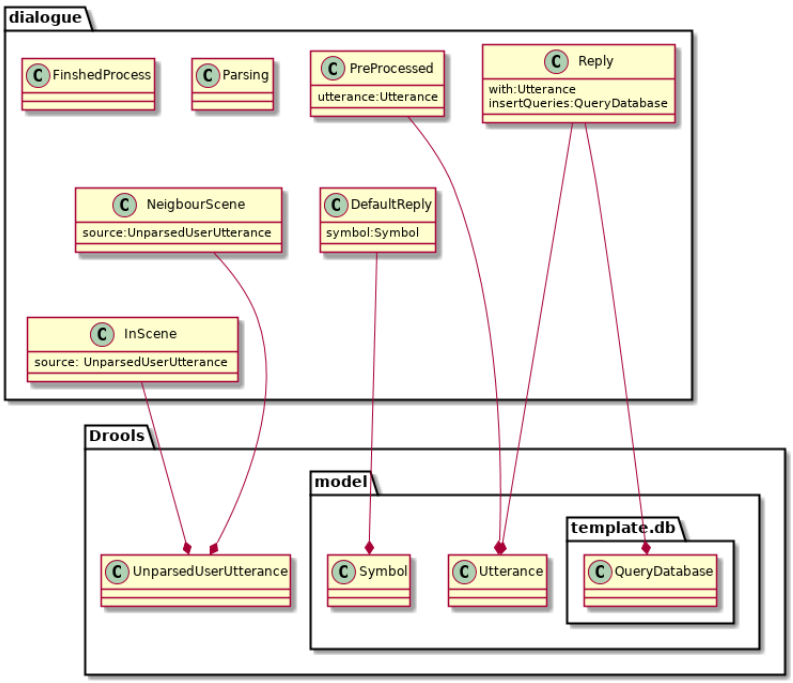


Figure 18: Supporting types in the dialogue Drools package

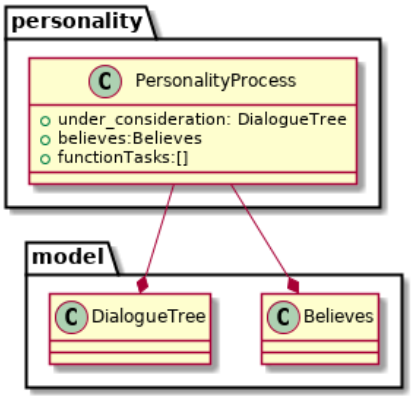


Figure 19: Supporting type in the personality Drools package

## 5.3 Initialization

Dealing with cyclic immutable data structures is a problem. If we were to store the connections in the nodes, and a cycle would occur, updating the first node would invalidate the second node. A way of working around this problem is by letting the connections point to an address of the node, rather than the object itself. Another way of working around such a problem is having a mutable, cooking phase, and after that make the object immutable [114]. This is in essence what we do with the `Database` structure. We construct its data first with a standard java `HashMap`, and once this is complete we wrap this into the `Database` class. Which makes a shallow copy and has no API for mutation (see figure 14).

This initialization problem is the reason why we chose the order of initialization shown in figure 20. The cyclic structure we want to create is the symbol graph $G$. So we start with the nodes in the graph, which are the symbols by constructing the symbol database. All symbols are constructed and put into the symbol database with as key a string containing the scene name and symbol name.

Once we have the symbol database we can use it to create connections from it. We can see in figure 14 that a connection database consists of symbol key values leading up to a set of connections. In figure 12 we see that connections consist of a symbol object it's going too with some additional values. To get the symbol values we just do a lookup in the symbol database, we know the key value from the file system and the `_connections` YAML file. In an initial iteration the symbol object wasn't used directly as key, instead the symbol scene name and symbol name string were used. However using the symbol directly is more type safe and ergonomic. The other values of the connection class can be determined by the YAML as discussed in section 6.4. An alternative approach would have been to store connections inside the symbols themselves, however this would make it impossible for the symbols to be immutable.

Finally the pattern databases are constructed. Patterns were after reading the symbols already put into a `HasHmap`, with as key the symbol and as value the set of patterns. So the only things that needs to happen for the in scene patterns database is to group them by scene. For neighbouring scene patterns however connections are required as discussed in section 5.2.3. This is why we postpone constructing these to the end.

## 5.4 Operation

To understand how the operation of a bot works, we can look at it from the point when a message is received and walk trough the steps it takes. An outline of this process is giving in the figure 21. Together with the outline and the figures defined in section 5.2.6, we can quite precisely explain what is going on.

This operation is starkly different from the one presented in section 2.6.4, particularly if you compare the activity diagrams in figure 8 with figure 21. What we can see directly by comparing these is the change in swimming lanes [115]. The Alice swimming lane has been removed completely, and in its place we've got Drools, which has become the center of the application. Then the Personality swimming lane was introduced, this is of course in light of this thesis.

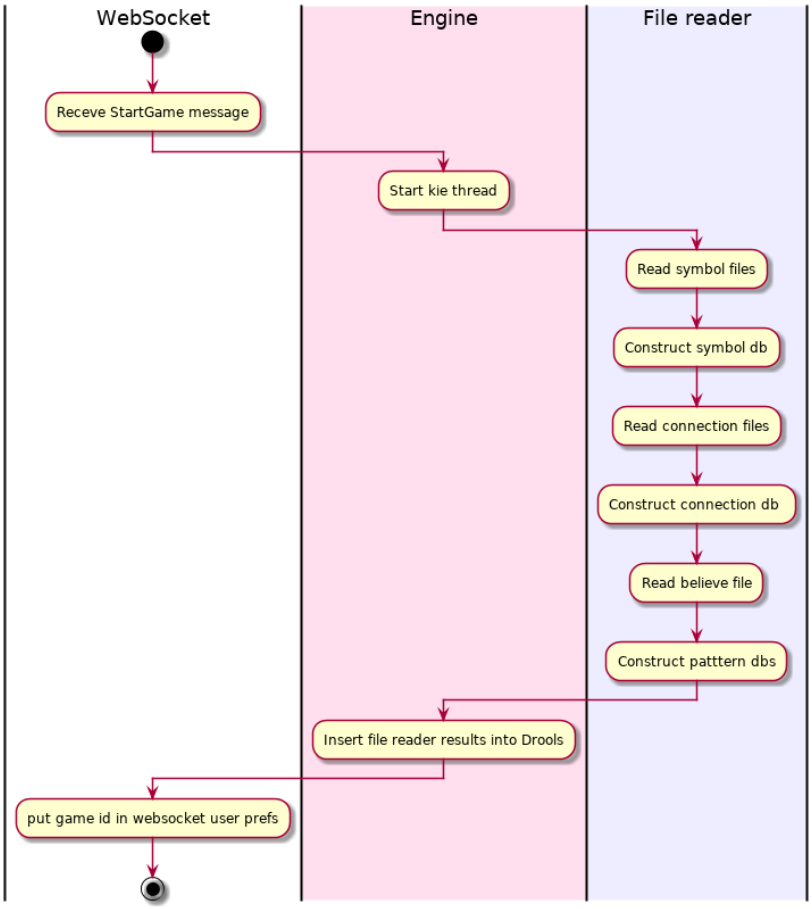


Figure 20: Activity diagram of a server game construction

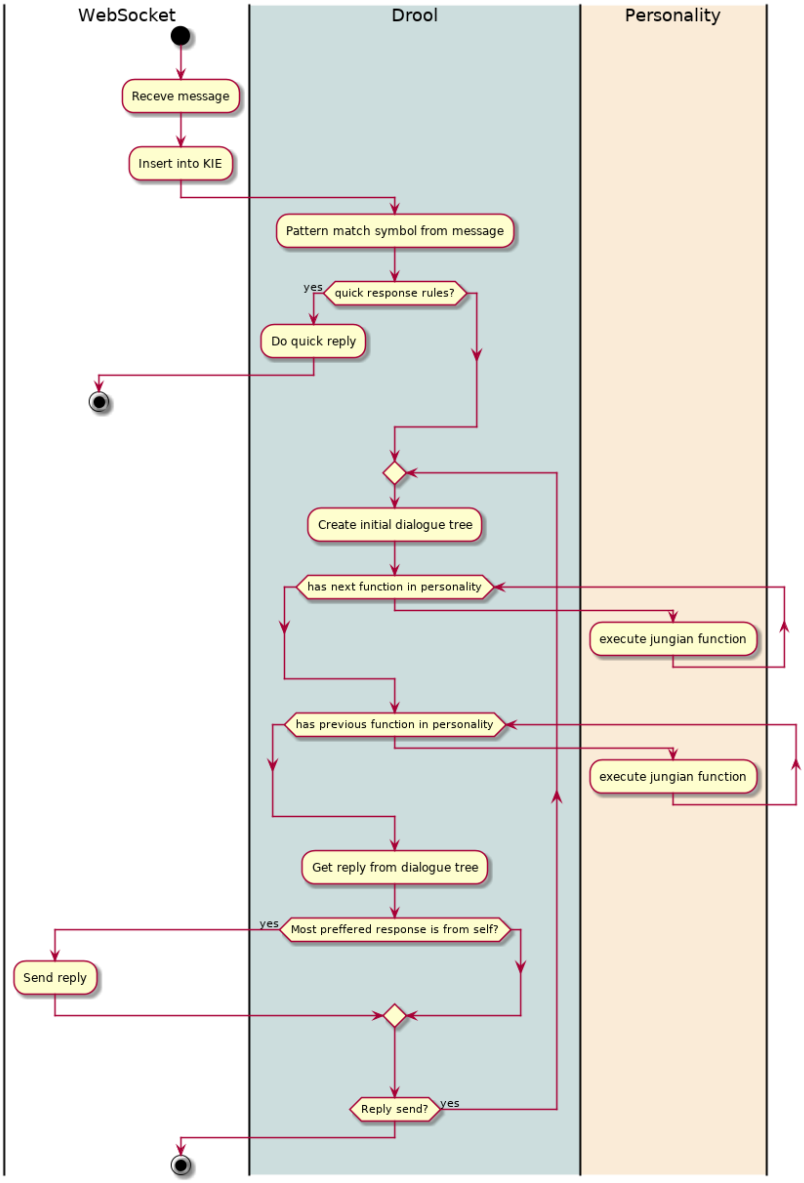


Figure 21: Activity diagram of deliberating on a user message

Since these activity diagrams are quite detailed in their description of what is going on, we made an overview of the key changes in the state diagrams presented in figure 9 and figure 22. In the new architecture, everything happens inside Drools. Only technical things such as dealing with the protocol and setting up the connection are handled outside Drools. This makes it impossible to bypass it, and it also opens up more space to do high level deliberations. Finally since `PatternMatching` is just the execution of another rule, we're not just limited to just pattern matching schemes in figuring out what the user said. Alternatives approaches are discussed in section 8.2.5.

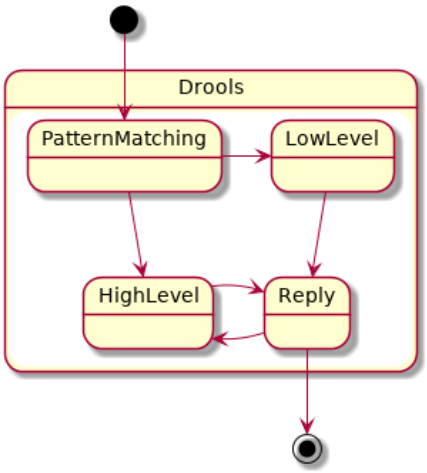


Figure 22: State diagram: Utterance processing with Drools

To ensure rules are executed in a particular order we often wrap and unwrap required data into types, as explained in section 5.2.6. For example the initial user utterance gets wrapped into an `UnparsedUttarence` type, before it's even inserted into the Drools. This type can be seen in figure 17. We could have just inserted a string and not created the type, but the reason for doing this with the initial string is to make it explicit: *This string needs to be parsed.*

So in Drools we can match on this type (see section 2.2.4). Which we do in the next step, parsing this string with pattern (regex) matching. This is actually the $\sigma \rightarrow s$ operation from section 4.3.1. To do this we use the pattern databases from figure 14, on which we use the active scene as a key (which is stored in the fact base) and then just match against all from the resulting set, this results in the `ParsedUtterance` type which can be seen in figure 17. This type then gets inserted into the fact base to continue the process.

The `ParsedUtterance` then gets transformed into a `PreProcessed` type. During this process the duplicates matches are removed. Each element in the `ParsedUtterance` list gets individually inserted as a `PreProcessed` type into the fact base. The reason for this in-between step is because we don't know how to handle multiple matches. So we just insert every uniquely matched item. In contrast to the initial approach where only the first match was used, this approach is more flexible. It allows the bot to form opinions about utterance where it doesn't necessarily wants to reply upon. For example if it asks the doctor "How are you doing?" the answer of "I'm good, how can I help you?"

or "how can I help you?" should be treated differently. It now can also give multiple answers to longer user utterances. However the disadvantage is that sometimes the bot will give more replies than desired.

As a first priority the low level reply rules can be fired. What they do when fired is removing the `PreProcessed` type, so that the high level rules don't get a chance to fire. This is modelled in figure 21 as an if else branch, which is true in practice, but no concrete if else structure is used. Drools has support for setting priority of execution in rules, which was used for this.

It should be noted that at the point of quick reply personality could also be at play. For example people could have alternative ways of pronouncing the response. Thinking people may for example respond with a confident yes, whereas feeling people would say it by default in a more doubting tone. We have not taken such variations into consideration.

The high level processing executes if there is still a `PreProcessed` type available, in other words no low level replies were executed. We create the initial `DialogueTree`, and remove the `Believes` from the fact base and put these believes into a `PersonalityProcess` which can be seen in figure 19. The reason for removing the believes base is to prevent concurrent modifications, by removing the `Believes` structure, rules that use it are no longer executed. To create a `PersonalityProcess`, a `Believes` structure has to be available. In this Personality process we also add the `JungianFunction` list, these are the functions that are extensively described in section 4.4, and its Java adaptation is described in figure 15. With this list it is determined which function should be executed next upon the `DialogueTree` and `Believes`.

After there are no more functions in the list, we know we are done. We move to the next step where we get the reply from the `DialogueTree`. This is an `Utterance` structure, if this `Utterance` is the same as the self field in the `Believes` structure, we send the reply by wrapping the utterance in a `Reply` type. If we don't send a reply, we insert a `FinishedProcess` type. If we do send a reply we reinsert the selected `Utterance` as a `PreProcessed` type. These types can be found in figure 18.

## 5.5 Social practice support

Currently our support for social practice is rather limited. It was not a core goal of this thesis to support this, however it was important that in future work it should be possible to add this. This was a reason to keep using a rule engine as core deliberation mechanism. We tried to make the deliberation process as transparent as possible to the rule engine.

Because this entire process is implemented in Drools, and we use types to track progress. It's relatively easy to add other rules that can modify the process, without changing the existing ones. Priority can be used to intercept a rule, as was done with the low level replies. Adding a more refined implementation of social practice therefore would be relatively easy. There exists already some support for the social practice in Drools, for example the scenes logic, but this is not complete. Better support can easily be added by adding more rules and tweaking with priorities.

Besides using extra rules to add support for social practice logic, for the personality part of the thesis specifically there is another possibility. They can be wrapped in a social practice function, that analyzes the result of the

personality function and then does social practice operations to the resulting `DialogueTree` or `Believes`. So based upon the social practice, and the personality function things may change.

## 5.6 Multilogue architecture

The architecture presented in section 5.1 is for a dialogue game. However a social practice does not put limits on the amount of participants, so what we really want is a multilogue architecture. Since the presented architecture in section 5.1 is relatively close to that we shall discuss here how to finish it. What we therefore will discuss in this section is the required changes to make it a true multilogue architecture, and thereby making it easier to implement social practice theory. Sadly there was no time to do the actual implementation of such an architecture. A deployment diagram of this architecture can bee seen in figure 23.

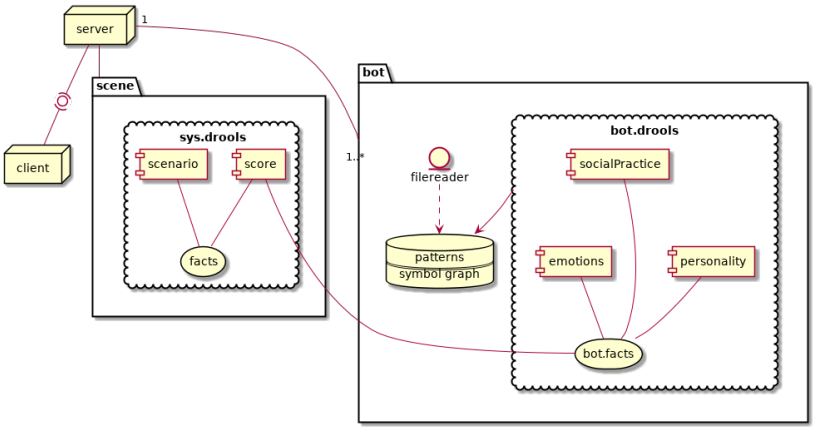


Figure 23: Deployment diagram of desired architecture

If we compare figure 23 with figure 11 from section 5.1, we can see several large changes. First of all the scene and scenario are now split of from the bot Drools. The reason for this is discussed in section 5.6.1, but it is basically to separate the system from the agent believes. Also note that there is just one system, but there can be multiple bots. Each of these bots has its own file reader, pattern database, and symbol graph. We introduced the social practice component as a way for the bots to predict how the scenario will go. The score component has direct access to the bot facts to evaluate how the user is doing, for example by analyzing the bots' emotions.

### 5.6.1 System vs agent believes

What is required of the Drools is that we make a separation between the multilogue *system* and agent *believes*. A good step in this direction is the `Believes` structure, which groups most agent thoughts, at least those used by the high-level system. Although it should be noted that the `Believes` structure itself also has problems, this is discussed in section 5.2.2, but the gist of it is that it's better to replace this structure with a Drools fact base. Since it has a self

field however, it could be used to identify an agents' believes. In other parts of the current architecture this is not the case at all. For example the `PatternDatabase` are just plainly inserted into the fact base. Which means we can't identify who's patterns these are.

The naive solution is to just mark every fact with a self field. Aside from the fact that you now introduce boilerplate code [116], this has another more serious problem, it grands the ability for agents to read each others minds. Since every believe structure, and thus agent, will live in the same fact base. This is of course not desirable, it would defeat the entire purpose of a multilogue system, at least for the agents.

So what we present in figure 23 is a different approach. What we do is separate the system from the agent believes. The system will be in the `sys.drools` engine, whereas the bot will be in the `bot.drools`. These are separate KIE runtimes, but the system has access to the bot.drools facts so it can do scoring and actually send responses for the bot. The server can then have multiple bot instances. In these instances not every fact has to be marked since they have their own fact base anyway. The only thing that needs happen is a self fact needs to be inserted, or the `Believes` structure could be used for that, if it is kept around.

### 5.6.2 Identifying speaker

Another big issue is figuring out which agent(s) is being talked to by the user. There has been some research on this subject, for example using hand coded selection mechanisms [117], and statistical mechanisms [118].

Although both these approaches are good options for robots, we are dealing with an application, in which we can cheat to get basically nearly 100% correct selection. We can simply modify the user interface, to let the user select which agent he talks too. Then the only thing that needs to be done is modifying the protocol so that the selected agent(s) get the message. You can of course extend this to select groups of agents, or make the selection based on some kind of abstraction, such as the distinction between whisper to an agent or shout to an agent.

In the data structures however a more structural change needs to occur, namely in both the connections and the utterances. The `Utterance` class need to be extended with to who the utterance was said, and the `Connection` class needs to get an additional restriction on to whom this connection can be said too. These definitions can of course quickly get out of hand, especially with larger groups, and therefore it would be wise to define some kind of role mechanism, for which solutions exist [119].

### 5.6.3 Practicalities

Then there are some practicalities to consider, that are more at implementation level. For example replies should be timed, so that the user won't see suddenly 200 lines of chat messages between two of his conversation partners. This is relatively complex to do because with multiple bots and doing timed reactions some kind of communication needs to occur between the bots. This is probably a task for the system, but it could also be considered a part of the social practice. Do note that these timed replies should be interruptible by the user. A different

way of dealing with this is just preventing the scenario from continuing at certain key points until a user reaction occurs. Deciding which approach is the best is a matter of experimentation.

Then it should also be noted that the scenarios of the bots should be specified differently. At least for parts of it, while other parts should be shareable. So the YAML files should be specified distinctly per bot, the Drools files distinctly per bot and there could be a mechanism to share these.

# 6 Replacing AIML

In this chapter we will discuss AIML in detail, we will explain why AIML doesn't fit our requirement. Then we will start analyzing the structure of AIML and see how we can alter it to make it fit our requirements better. After that we will introduce a new modeling system, that is more succinct and flexible then AIML. Finally we will present a script to convert from AIML to the new format.

## 6.1 AIML issues

Originally, AIML was used for mapping user input to a reply. However, as explained in section 5, AIML has some problems for our particular use case. The major two issues are:

- No way of giving Drools space to do deliberation aside from updating the scenario.
- No way of accessing the knowledge base from inside Drools.

A relatively simple solution to both of these would be to use an XML parser to load AIML into a data structure and insert it into Drools, however this won't solve the core of our problem: The way AIML models a conversation makes it hard if not impossible to list possible actions for an agent at a particular point in a conversation, therefore no true deliberation is ever possible as long as AIML is used. As we will see in the coming paragraphs, trying to adapt AIML to fit the requirements will transform AIML into something else.

To give a good intuition of this issue, we use an example. Listing 4 shows a piece of dialogue modelled in AIML. This is a pretty standard piece of AIML, no surprises there. Patterns are used to identify user utterances and attach responses to them. We created a deployment diagram of the situation seen in figure 24, which removes the syntax, so that the situation is more obvious.

What we can see is that if the user says "Hello" then the bot will reply "Hi". So what we have here is a mapping function from $\sigma \rightarrow \sigma$, where $\sigma$ is a string (see section 4.3.1). The issue is that once Hello is matched, the answer **must** be Hi. S-AIML extends this with adding drool tags, but these are inside the template tags and therefore cannot truly get out of this relationship. Unless they were to replace the entire content, in which case AIML is no longer used as knowledge representation anyway.

We're also not modelling the entire conversation. There is no way for the bot of knowing that his Hi utterance, could be followed up reasonably with the question "How are you?" for example. Therefore there is no way for the bot to plan ahead in the conversation, or have any kind of variations in these plans. This is of course a problem, because we imagine personality as variations in a process, or plans (See section 2.2.5).

There are also some other small problems, which aren't really that important for this thesis in particular, but worth mentioning. Such as with AIML one has to use the AIML based patterns defined by the standard [120], there is no way to use full regular expressions, or statistical methods. Another issue that we can't access encoded utterances directly, so from Drools there is no way to do spontaneous utterances unless they're hard coded strings. Finally, since Drools can be skipped all together (see figure 9), the bot could have amnesia about

```
<aiml>
    <categroy>
        <pattern>
           Hello
        </pattern>
        <template>
           Hi
        </template>
    </categroy>
    <category>
        <pattern>
            How are you
        </pattern>
        <template>
            Not doing too well today.
        </template>
    </category>
    <category>
        <pattern>
            How * you
        </pattern>
        <template>
            <srai>How are you</srai>
        </template>
    </category>
</aiml>
```

Listing 4: Standard AIML categories

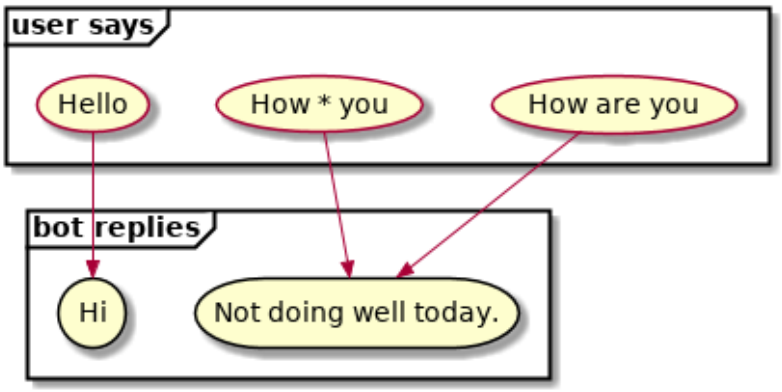


Figure 24: Deployment diagram of AIML example

certain utterances, to prevent this every template tag would need an insert tag, which is boilerplate code [116].

What needs to happen is either to either modify the Alice bot and AIML to work fundamentally different, or outright replace it with something else. We chose for the latter, we did so by first carefully analyzing AIML and deciding which parts we want to keep, and which other parts we wanted to remove or change. The next sections will discuss the thought process to a new representation.

## 6.2 Analyzing AIML

What we want to do is create a mapping function $\sigma \xrightarrow{g} s$ (see section 4.3.1). In our first attempt we will modify AIML to do this. AIML is primarily a case based reasoner. It will match on predefined strings or patterns and then 'say' the string that was attached to the pattern. So we can use AIML to match user input, but the language has to be modified so that instead of producing a reaction, it will indicate what the symbol is that was matched. An example of these changes can be seen if we compare the listing 4 from the previous section with listing 5. The new deployment diagram can be seen in figure 25.

```
<aiml>
    <categroy>
        <pattern>
           Hello
        </pattern>
        <symbol>
          Greeting
        </symbol>
    </categroy>
    <category>
        <pattern>
            How are you
        </pattern>
        <symbol>
          StatusInquiry
        </symbol>
    </category>
    <category>
        <pattern>
            How * you
        </pattern>
        <symbol>
          StatusInquiry
        </symbol>
    </category>
</aiml>
```

Listing 5: AIML that refers to 'symbols' rather than templates.

These changes remove the reactive nature of the chatbot, no longer do patterns indicate what to reply to, but instead simply what they are. This example can't be functional of course, since the symbol graph hasn't been introduced (see section 4.3.3), so there are no replies. However we can at least use these to to create the symbol graph from.

With this we almost have solved the first problem of not being able to do true deliberation. Of course we can't make a reply of this yet since we still have the second problem to deal with, how to access information stored in these symbols. In other words, dealing with the $g'$ function, this is what the next section deals with.

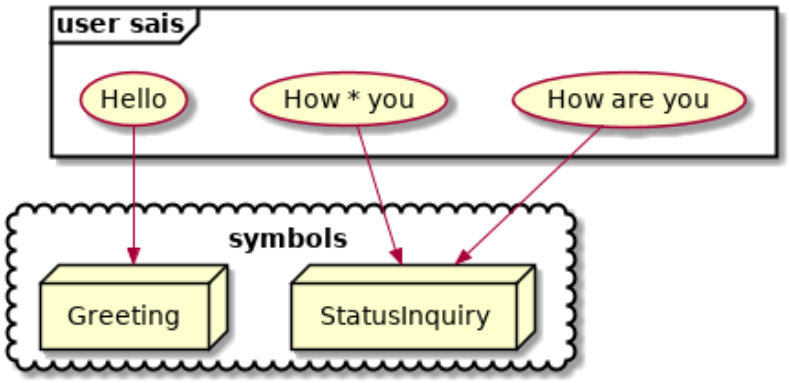


Figure 25: Patterns to symbols

### 6.2.1 Symbol to string

The next step is to consider how we go from symbols back to strings. What we assume is that the agent already found a symbol to utter, for example the status inquiry. This string from a status inquiry is already available in AIML, they were called template tags. We just need to separate the literal strings from the catch all patterns as can be seen in listing 6. In this listing we renamed the template tag to literal, because the name 'template' is very generic and we have already used it for another system discussed in section 5.2.5 and 6.5. They function the same: Provide the string to utter for that particular category, in other words the $s \xrightarrow{g'} \sigma$ function (see section 4.3.1).

In deployment diagram 26 we can see how this works in memory. Patterns point to symbols, which have the literal utterance available to them. Since the names of symbols are not the same as the literal content, we can use terse descriptive names for more verbose content. Such a property is useful for referring to the symbols from inside the drool engine.

```
<aiml>
    <categroy>
        <literal>
           Hello
        </literal>
        <symbol>
          Greeting
        </symbol>
    </categroy>
    <category>
        <literal>
            How are you?
        </literal>
        <patterns>
            <pattern>How * you</pattern>
            <pattern>How are you *</pattern>
        </patterns>
        <symbol>
          StatusInquiry
        </symbol>
    </category>
</aiml>
```

Listing 6: AIML with grouped patterns and string literals

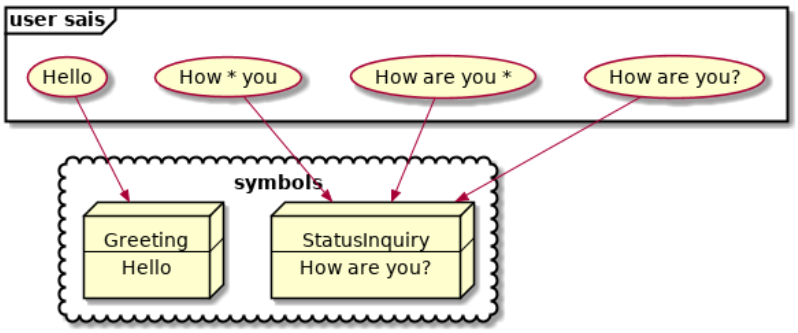


Figure 26: Patterns to symbols with literals

We can make the syntax from listing 6 even more terse. Observe how one category always will have one symbol tag. If we were to extract the value of this tag and use it as a filename, we can ensure that each symbol is only declared once. Then we can also assume that the AIML tags and category tags are implicit. This results a terse definition as can be seen in listing 7. This doesn't include the `Hello` code because that was part of another symbol, and therefore another file.

```
<literal>
    How are you?
</literal>
<patterns>
    <pattern>How * you</pattern>
    <pattern>How are you *</pattern>
</patterns>
```

Listing 7: Terse AIML but illegal XML

However the observant reader will see this isn't valid XML and by extension AIML, since XML requires a single document root tag. It does specify what we want, and it does so very tersely. Since we are changing the semantics of AIML drastically we may as well use a more terse data format in which such a definition is legal. In the next section we will analyze some possible candidates.

### 6.2.2 XML vs JSON vs YAML, vs TOML

All these languages are standards [121, 122, 123, 124]. However there is a major difference between XML and JSON or YAML, and that is their intention. XML is a markup language, whereas both JSON and YAML are data formats [125]. What they all share is that they are supposed to be both human readable and parsable by a computer program.

So what we mean by human and program readable is that with relatively little effort, a human or program can understand what's going on. Unlike say a binary format, which first needs to be parsed by a program before a human can understand it (or with great amount of effort). Alternatively unlike a human document, such as a book, a computer program needs to do lots of effort to extract information from that, whereas a human can just understand it 'naturally'.

Markup languages provide a nice middle ground between human readable and program readable. XML does this by inserting tags (metadata, which are usually just annotated words), that gives meaning for the program, so that authors can focus on the structure and content of the document [126].

In AIML however, no document is constructed. It's not a story with a lead, middle ground and conclusion, it's more like a key-value pair database or big configuration file. So since XML was intended to be used for inline document markup, We will argue that there are better alternatives that focus on key-value based configuration in particular, such as YAML, JSON or TOML.

JSON is by far the best known format of these three remaining contenders. However JSON has a few significant disadvantages, for one it doesn't allow comments and it is quite strict. For example a common acceptable mistake is to have an array with a trailing comma like: `[1,23,4,]`. This is an error in JSON [127], while it's relatively easy for a program to notice this is erroneous and therefore correct it. The syntax of JSON can be also more terse, which is shown by both YAML and TOML. Because of these reason we looked at the other two contenders.

The reason for choosing YAML over TOML is that YAML is stable, whereas TOML hasn't reached stabilization yet [128]. Although the argument TOML has over YAML is that YAML is much more complex, and can in certain cases be not as human readable [129]. If TOML were to stabilize, it's probably better to move to that format instead. Its instability means that the author thinks the representation may change. Therefore currently the best choice is YAML. It's relatively easy to move between these standards, if they're stable, as is shown in section 6.6.4. Therefore a possible future move from YAML to TOML should also be easy. Note that because we separated file reading from the main bot we could even support multiple data formats (see section 5.1).

## 6.3 Using YAML

It was decided conclusively to use YAML instead of AIML for symbol representation. How these symbols are represented in YAML can be seen in listing 8. This is only a syntactic change, therefore no deployment diagram is made of this.

Rather than having a literal tag open and close to show what the literal is, YAML uses a key value structure. This key value structure is either a dictionary (in Java `HashMap`), or a class (see section 6.3.1). Lists can be indicated by using dashes in front of each item. Note that we still use the idea of having one symbol per file, and the symbol name is the filename. Therefore no `symbol` item is shown in the example. Quotation marks `"`, are to indicate strings, for certain characters this is necessary, in other cases YAML does this for the author.

Note that there are multiple literals to choose from in listing 8. In the current implementation this is similar to using `random` tags in AIML. However as can be seen in section 5.2.1, this list of strings is just available to the implementation. Therefore social practice could use it for example, making the selection situational, and this selection could be influenced by personality.

We keep the pattern fields for legacy support. However it's much more easy to just support regular expressions [130]. The reason for this is that they're part of the Java standard library [131]. What we do is transform AIML patterns into regular expressions, and then use the Java standard library to match

```yaml
literals:
  - How are you
  - "What's up?"
patterns:
  - How * you
  - How are you *
```

Listing 8: YAML symbol representation

the patterns. This will make the patterns more precise, and give less code to maintain, an example can be seen in listing 9.

```yaml
literals:
  - How are you
regexes:
  - "How ([a-z])\\w+ you(.*)"
```

Listing 9: YAML with regular expressions

The example in listing 9 only matches a single word and allows for trailing characters. Note that the example in listing 8 would've also matched "How did this became you?", since any character would've matched the star (not any but most characters, see section 6.5.1). Regular expressions can be much more precise in specifying what a wildcard star is, although they are also a lot more difficult to learn and read. Therefore both methods will be supported.

### 6.3.1 Loading YAML

As explained earlier YAML key value pairs are either directly loaded into a `HashMap`, or into a class. We chose to use classes wherever possible because it provides more type safety. This is only possible if all keys are known before hand. Which it's not in case of the template system for example, but in most other cases it is.

In listing 10 the class where the symbols are loaded into is shown. The fields defined there can be used as keys in symbol files. Note that over some fields the user has no control, such as `name` and `scene`. Other fields such as `literal` need to have at least one value, although not shown in this structure. There are systems that allow declaration of fields and restrictions upon them to be shared [132], which is more convenient for the user, but we didn't do that in the interest of time.

```java
public class RawSymbol {
    public String name; // defined implicitly by filename, user can't override
    public String scene; // defined implicitly by folder, user can't override
    public List<String> literals = new ArrayList<>(1);
    public List<String> patterns = new ArrayList<>(1);
    public List<String> regexes = new ArrayList<>(1);
    ...
}
```

Listing 10: Class that deals with symbol files

In listing 11 we show how one can load a YAML structure into a raw class. We also add some additional restrictions, we do not want to load multiple objects from a symbol file, although YAML supports it. This would break the name uniqueness guarantee the file system provides.

```
List<RawSymbol> syms = YapFactory.readAsYML(file, RawSymbol.class);
if(syms.size() > 1){
    throw new RuntimeException("Can't deal with multiple yml " +
            "objects in symbol file, please use different files " +
            "for symbols so that their names are gauranteed to " +
            "be unique, error in " + file.getName().getBaseName());
}
```

Listing 11: Reading with the class from listing 10

## 6.4 Connections

Now we've figured out how to represent our basic axioms, the symbols thoroughly, we can move on to make our model functional again. So what we have lost in the process are our update rules (see section 2.5). In section 4.3.3 we model these as the symbol graph. However we have not made a syntactic representation of this, which will be done in this section.

### 6.4.1 Core idea

What we do with the symbol graph is re-adding the implicit connection from AIML between patterns and templates. Both patterns and templates have now become plain symbols (see section 6.6.2). If you just have symbols the bot won't know what to do when one of these is inserted without extra information. This is what we represent with connections. What sensible replies can we give if a symbol is uttered.

We will show how this works with help of an example. In listing 12 we connected the symbols discussed in our original example from section 6.2. With these connections we can see the situation illustrated as deployment diagram 27, From this we can see that a `greeting` can be replied to with a `greeting` or with a `status_inquery`. Because we had multiple patterns leading up to these symbols, we now have multiple responses to multiple patterns. This means there is a choice now once a `greeting` is uttered, and choice opens up room for personality as a process.

```
from:
 - greeting
to:
 - symbol: status_inquery
 - symbol: greeting
```

Listing 12: Connections grouped into a file

### 6.4.2 Single file

Originally we just wanted to stick to the (implicit) AIML approach of connections, and add them within the symbol files as can be seen in listing 13.

```
literal: Why are you here
to:
  - need_medicine
  - broken_arms
  - feel_sick
```

Listing 13: Intrusive connections

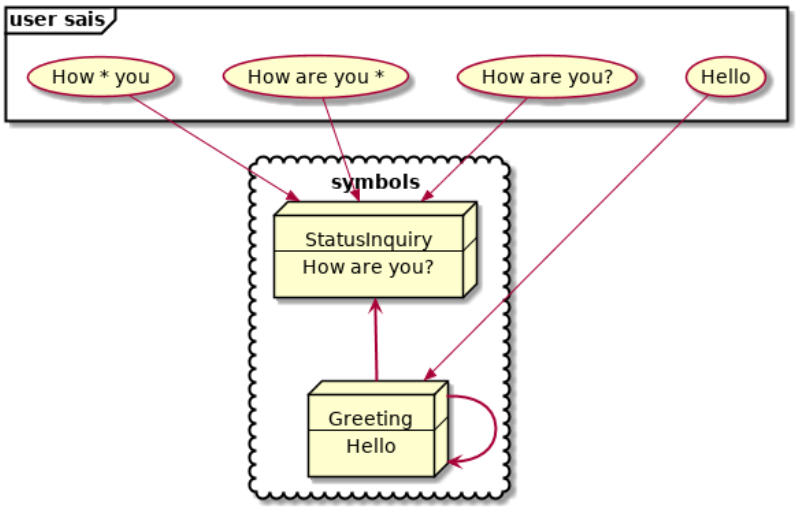


Figure 27: Symbol graph deployment diagram

However we decided against such an approach. There were two reasons for this, the first one is that we found out that connections are much more complex than simple one directional links. They for example also need to contain perlocutionary values and be marked with which agent can use the connection (default being all agents). The second reason is a practical concern, it turns out that if you group all connections into a single file you can get a better overview of which connections are made. So you get a better sense of what could be said when. What we do now is a grouping of connections into their own special file.

Take as an example listing 14 which is a connections file of a scene. In figure 28 we can see a deployment diagram of that connections file. This is not a complete conversation of course just an expert of a larger model. Note that reading the code from listing 14 is not much harder to understand than the deployment diagram. We can see that the connections modeled here are already quite complex, as this file grows in size the dialogue becomes harder to manage. Usage of scenes and tools in text editors such as searching become quite vital. In any case it should be less of an issue than in AIML since this syntax is more terse.

**Actors** A foreign idea to AIML is the notion of actor restrictions. Since AIML will always model from the perspective of the bot, there is no need for this. However because we model both bot and user, we do require this.

To make sure certain strange situations don't occur, such as the patient asking the doctor if the doctor is sick, we added actor based restrictions on connections. This is done trough the `restricted_to` key, which can be seen in listing 14. Currently however this implementation is rather limited, it only allows for an 'any' actor, which basically means any actor can say this, or a specific actor. This could be extended with some kind of role system [119].

Note that because AIML does not specify actors, it can never be used on its own to model a multilogue system in. Although we make actors explicit not

```yaml
from:
 - greeting
to:
 - symbol: greeting
 - symbol: ask_reason_here
   restricted_to: doctor
---
from:
 - ask_reason_here
to:
 - restricted_to: patient
   symbol: need_medicine
 - restricted_to: patient
   symbol: broken_arms
 - restricted_to: patient
   symbol: feel_sick
---
from:
 - need_medicine
 - greeting
to:
 - restricted_to: doctor
   symbol: why_need
 - symbol: status_inquery
```

Listing 14: Connections grouped into a file

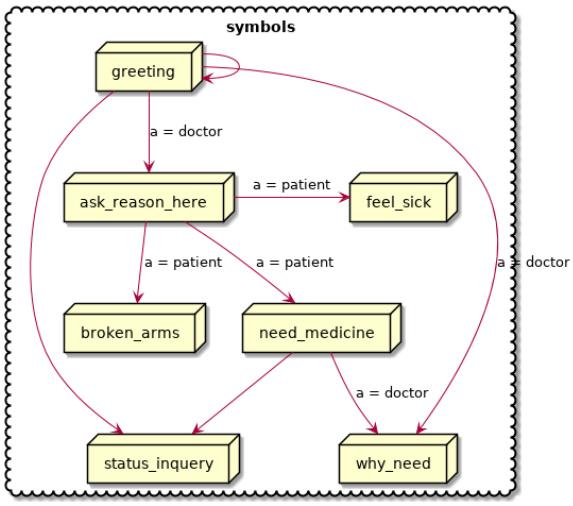


Figure 28: Symbol graph of connections grouped in file

because of the desire to implement a multilogue system, but to make it possible to predict what the other actor will say from the agent's perspective. Being able to eventually build a multilogue system on top of this is a nice benefit.

### 6.4.3 Scenes

In the S-AIML extension, scenes were used to enable and disable certain patterns. Unlike the topic mechanism of AIML which just gives preference, something from another scene wouldn't be used at all. This is makes it easier to write dialogues, because we don't have to worry about patterns from other scenes, so we can use more generalized patterns in our particular scene.

Although scenes aren't necessary to implement the theory presented in section 4, it is necessary for eventual social practice support. It also makes this representation much more modular. The part were scenes are actively used is in section 5.2.3, patterns are grouped based upon scene in the `PatternDatabase`.

The way we represent scenes is rather simple. We use directories as scenes, all the symbols within a directory are in that scene. Connections are implicitly assumed to be between symbols in the same scene as where the connection file is in, unless stated otherwise, which can be seen for example in listing 15.

```
1  from:
2   - status
3  to:
4   - symbol: ask_reason_here
5     scene: information_gathering
6     restricted_to: doctor
7   - symbol: good
8     restricted_to: doctor
```

Listing 15: Scene example in connections

In listing 15 we can see two connections being modelled, from the `status` symbol to `ask_reason_here` symbol that transitions to a new scene, and from the `status` symbol to the `good` symbol that stays in the same scene.

We break away from the idea of S-AIML that scenes are linear. We believe that scenes are a weakly connected cyclic graphs. Note that by cyclic, we mean there can be cycles, but there don't have to be. So we can go back to previous visited scenes for example. It's weakly connected because it wouldn't make much sense to model scenes that can't get to other scenes trough connections. Although there is nothing preventing one from doing so, the common sense reasoning is: Why would you model a scene that is inaccessible from the rest of your model?

### 6.4.4 That tags

The 'that' tags in AIML have a rather unfortunate name. They require the bot to have said something before the current pattern can be used. Therefore in our YAML semantics, they are an additional restriction on connections. They appear in listing 16 as an example.

Although supporting this feature isn't necessary to implement the theory presented in section 4, they do allow much more fine grained control over the scenario. We therefore decided to also implement this AIML feature.

'That' tags act as an extra filter upon the category. Before this category becomes active, the bot first has to have uttered the pattern in the 'that' tags.

```
<category>
    <that>Why is doctor Aarts not here I am one of his patients.</that>
    <pattern>
        surprised you're doctor
    </pattern>
    <template>
        <insert packageName="scenarios.large.global" typeName="SentenceSpoken" />
        <insert packageName="scenarios.large.prehistory" typeName="BadAndLateExplanation" />
        I did expect him to be here, yes.
    </template>
</category>
```

Listing 16: 'That' tags example.

In the new representation we can model such a thing too. To model 'that' tags we add an extra optional field to the `RawConnections` called 'before', with two required fields, 'who' and 'said'. The actor is indicated by 'who' and 'said' indicates what was said. An example can be seen in listing 17.

```
before:
   who: patient
   said: "why_is_docter"
from:
 - "surprised_your_doctor"
to:
 - symbol: "expect_him_be_here"
   restricted_to: patient
```

Listing 17: 'That' tags as before fields in YAML

Adding the who field was necessary since we no longer model just the utterances of the bot, we needed to expend it by adding the actor who uttered the utterance. We also needed to think about when it was uttered, because alternation isn't a guarantee in conversation (discussed in section 4.5.1). What is done now, is to filter out the utterances from the person who is not the who in the before tag, and then take the latest utterance from them.

To surpass the AIML 'that' tags, we can extend this mechanism by making the `RawBefore` type self recursive with an optional field before of its own type. This is best illustrated with an example which can be seen in listing 18.

```
before:
   who: patient
   said: "why_is_docter"
   before:
      who: doctor
      said: "imhe_doctor"
from:
 - "surprised_your_doctor"
to:
 - symbol: "expect_him_be_here"
   restricted_to: patient
```

Listing 18: Before recursion

The default value of the before field will be `None`. Which just means no additional restrictions. In this example making such an explicit definition won't be very constructive. However since we need an identity (no-op) before value anyway for the default restriction, making a recursive definition isn't a big extension.

The AIML tag specification never defined when a 'that' tag match would be active [**?** ]. However it can be derived implicitly from the reference [**?** ] that it

should always be on the previous utterance. We could add options for when an utterance was made, for example just now or any previous utterance. However in the interest of time we won't do this.

## 6.5 Templates

In AIML, star tags are template focused. They capture the content of a wildcard at a particular index and allow them to occur in the bots reply. An example can be seen in listing 19. This is a simple trick to make the bot more flexible, and since in AIML patterns and templates occur hand in hand, doing this is easy. In this system we have not such a close relationship (on purpose), and therefore it's more difficult to implement.

```
<category>
    <pattern>
        name is *
    </pattern>
    <template>
        Okay, <star index="1" />. Can we get started?
    </template>
</category>
```

Listing 19: Star tag usage example.

Although this feature isn't necessary for personality per se, it's a really important chatbot feature. It allows symbols to become more flexible, and connections more dynamic.

To implement these in our new system we first need to analyze what star tags are. Then we need to figure out how we can implement this in our matching system, and finally we need to add a template method to our YAML representation. The data structures involved are discussed in section 5.2.5. What is discussed in this section is the behavior and syntax.

### 6.5.1 AIML stars

The AIML standard specifies [120] stars as a one based index scheme. Wildcards are defined as any string in $L(N)$ where $L(N)$ is all normal words. This would imply that it would not include spaces or other special symbols. Since this definition is rather vague we did some experimentation on the example from listing 19. We tried to introduce ourselves to the bot with various names as can be seen in table 5.

| Inserted | What the bot accepted |
|---|---|
| `jap34! gee!@` | `jap34` |
| `jap gee23` | `jap gee23` |
| `jap_23_flap ddd dfadf` | `jap_23_flap ddd dfadf` |
| `blah *. blah` | `blah *` |
| `blah * ahah` | `blah * ahah` |

Table 5: Attempts at what would pass for 'normal' words according to Alice.

Quite arbitrarily some characters are accepted while others aren't, Alice happily accepts an underscore but an exclamation mark is to much. There are several ways to improve this system. Such as trying to add context to what the

star should be. In our example case we wanted to match on names, so we know that numbers shouldn't be allowed, or at least that it would be highly unusual. Social practice and norms could help with these distinctions.

We already improved on the star matching system by using regular expressions [130], which would allow scenario writers to be much more precise in what they want to match. However we want to also be able to store what is matched and access it later. The java regex API already provides a method of extracting information from wildcards trough something called groups [133]. An example of an regular expression that extracts data can be seen in listing 20. So the problem is no longer one of extraction, but just about organizing that what has been matched, and putting it in a template.

```
literal:
    - "My name is paul" # what the bot would say if symbol used
regexes:
    - "My name is (?<name>.*)" # the matching mechanism, store wildcard into name field
```

Listing 20: Extracting data with a regular expression.

### 6.5.2 Templates in symbols

To insert data into a symbol we need a templating mechanism. We found three template libraries in Java: Velocity [134], FreeMarker [135] and StringTemplate [136]. Because StringTemplate enforces strict model view separation, and therefore is much more maintainable and easier to understand [137], we chose to use StringTemplate, over the other two options.

Using that engine we can define a naive way of writing a symbol as can be seen in listing 21.

```
literal: "Hello, my name is <actor.name>"
```

Listing 21: Naive approach

In listing 21 there is a problem however, is it always the case that the current actor should be used if we use a query like "<actor.name>"? No because we can say something like: "Hey <actor.name>, can I borrow your pen?". Therefore we should reconsider selecting such things at the symbol level. A lot of context information is not available in symbol files. The best we can do is give them a name and leave it at that, as can be seen in listing 22. This won't add any restrictions on the inserted value. It's just a hole that can be filled up by an arbitrary string.

```
literal: "Hello, my name is <name>"
--- # another file
literal: "Can I borrow a <tool>"
```

Listing 22: Context unaware approach

How do we fill these templates? The issue we had with trying to fill these in the symbols was the lack of context. Context is mainly provided by connections, if we know who is saying "Hello, my name is <name>", we know what to fill in for the template 'name'. In this system context is provided by connections. An example of how to do this is given in listing 23, where the name is filled by

using a previous utterance. We use the match label to identify which matched item needs to be extracted.

```
from:
- my_name_is_x
to:
- symbol: hello_x
  restricted_to: patient
  utterances:
    name: # the template name to fill
      actor: doctor
      symbol: my_name_is_x
      scene: introduction
      match: name # group name to extract
```

Listing 23: Connection syntax for filling templates

There are several important things to note about this implementation. First of all it's a lot more flexible than AIML stars, since information can be retrieved from any previously uttered utterance, if a connection has a query that isn't satisfied, it is not available. Another advantage over AIML is that regular expressions allow much more precise input sanitation than wildcards. This could enforce input that is only numeric for example. Which in turn can be used within Drools as a query. Finally we preform checks to make sure that symbols with template names only have connections leading up to them that satisfy the template names. AIML didn't have to do this in the first place of course, since stars only acted per category.

A feature that is missing is retrieving drool facts from the knowledge base and inserting them into the template. However one can workaround this limitation by writing Drools for this, similarly to low level replies. A `MatchedQueryDB` can be constructed manually to fill with drool facts.

## 6.6 Automatic AIML to YAML

AIML is a standard [120]. Note that the spec says: "AIML shall be compatible with XML.", and XML is also a standard [123]. YAML happens to also be a standard format [122]. With all these standards, it's relatively easy to convert between them, because libraries exist for parsing the standard, and in turn for generating it.

### 6.6.1 Legacy AIML

An introduction of a new format is nice, but when this is done, all the work in the old format could become obsolete. This is obviously not desirable. There are several ways of circumventing this. First of all, one could add support of the legacy format to the code base. The second method is creating a script that will help along the way with conversion. We chose the second method because the first method will be much harder, as it will re-introduce the problems we had with AIML in the first place. The second method provides the user an opportunity to make their AIML comply to the new method. The script does it automatically for most frequent cases, however we still expect user intervention to test and complete conversion.

To make the conversion we need to point out some structural observations about differences between AIML and YAML. Firstly AIML works strictly from

the perspective of the bot. There is no deliberation about what the user is thinking. Therefore we can't model what we expect the doctor to say after a patient uttered something in the test scenario because this information isn't encoded. You may argue that the mechanism which could be used for this are the *that* tags. But they are just an additional restriction on the pattern matching, see section 6.4.4. Alternatively you count the injection of types (in S-AIML), but this isn't a formal encoding in AIML itself, but rather a way of informing Drools what's going on.

### 6.6.2 Mapping AIML

What we can do is extract the patterns, and their respective literals into symbols. For example we have the following category in listing 24. In this example there are in terms of YAML two symbols and a connection. The first symbol is pattern tag, the second symbol is the template tag excluding the insertions, and the connection is from the pattern to the template which is restricted to the patient. So from this example we can define the mapping result in listing 25, with its respective file names in comments above.

```
<category>
    <pattern>
        How long * pain
    </pattern>
    <template>
        <insert packageName="scenarios.large.global" typeName="SentenceSpoken" />
        <insert packageName="scenarios.large.timelapse" typeName="DurationPain" />
        For a while now
    </template>
</category>
```

Listing 24: AIML mapping example

```
# for_a_while_now.yml
literal: "For a while now"
---
# how_long_*_pain.yml
literal: "How long * pain"
---
# _connections.yml
from:
 - "for_a_while_now"
to:
 - symbol: "how_long_*_pain"
   restricted_to: patient
```

Listing 25: Listing 24 expressed in YAML, with the filenames in comments.

A lot of category elements simply mean to add a pattern to a symbol, for example listing 26. What we can do with the SRAI tags, if they exist, is simply checking if the symbol exists and then add the pattern to that symbol. If the symbol does not exists yet, we create it anyway, since the content of a SRAI tag is a pattern itself. In this case, we just add both the pattern and content to the pattern list.

### 6.6.3 Type insertion

After the conversion is finished, the bot will mostly work, with the added bonus that the dialogue will be more readable. However there are some caveats, espe-

```
<category>
    <pattern>
        How long * pain *
    </pattern>
    <template>
        <srai>How long * pain</srai>
    </template>
</category>
```

Listing 26: SRAI tag that adds a pattern to a symbol

cially in the previously introduced variant called of AIML called S-AIML.

In S-AIML types are injected to track progress of the scenario. In contrast to the current scheme where the entire user utterance gets inserted and it's up to Drools to make a symbolic understanding from it. To work around the issue of inserting types we actually have to generate drools in case of a particular symbol inserted, which then inserts the type specified by the insert tag. For this we can just use the low level reply mechanism, but rather than replying we insert the specified type. This will allow the original drool rules to still execute. An example of this can be seen in listing 27.

```
rule "insert types when symbol how_long_pain was uttered"
when
    $pre:PreProcessed($symbol:utterance.what, $symbol.name == "timelapse/how_long_pain")
then
    log.info(drools.getRule().getName());

    scenarios.large.global.SentenceSpoken obj1 = new scenarios.large.global.SentenceSpoken();
    insert(obj1);

    scenarios.large.timelapse.DurationPain obj2 = new scenarios.large.timelapse.DurationPain();
    insert(obj2);

end
```

Listing 27: Generated Drools from listing 24 insert tags

Something which complicates this is that attributes can be set trough JSON within the insert tags. This can be handled inside the Drools rules because the inserted types always have setters. So there could be java code generated for that. Although we haven't done this in the actual implementation since this feature didn't seemed to be used a lot and we expect from the user to do manual modification of the generated results.

### 6.6.4 Proof of concept

To show that it is indeed easy to write a conversion script, with above restrictions kept in mind we made a proof of concept. This conversion script is located in the conversion folder from the root project of the git project. It is expected to be executed from command line and provides several command line arguments that are documented in the script itself.

The usage of the script can be shown by passing the help parameter to it. As can be seen in listing 28. There are some features missing from this script however. The help message will display which.

```
python main.py --help
```

Listing 28: Print conversion script usage

# 7 Implementation

In this chapter we will discuss the specific personality implementation details. We will discuss how to setup a scenario, and how we tested our system. In section 6 a lot of implementation specifics were already discussed.

In this chapter we will first setup a small case study of personality influence in section 7.1. Then we will show how to model a part of this scenario in section 7.2, to show how the bot works and how to steer it. Finally we will discuss how we tested the bot in section 7.3.

## 7.1 Personality influence case study

To make the influence of personality more concrete, we make a scenario of a doctor appointment where each patient has different personalities. First we have Sander the INTJ, secondly Susie the ENFP and Chris the ISTP. This type selection will give a rough usage of most Jungian functions. In all cases the patients have the same problem, a back pain. The cause of this problem in all cases is a worn out back. After the dialogue we will also discuss the motivations for saying things the way they do.

### 7.1.1 Sander the INTJ

First we should note the dominant and auxiliary functions of someone with an INTJ MBTI type. An INTJ has as dominant function introverted intuition $N_i$ and as auxiliary thinking extroverted $T_e$. We would expect these function to be most obvious in the dialogue (as discussed in section 2.1.2). $N_i$ mainly focuses on connecting ideas and extroverted analyses objects in the external world. Combined with each other we get a personality that focuses on getting to goals by analyzing the situation far ahead of time. This results in the expected dialogue which can be seen in table 6.

| Who | Utterance |
|---|---|
| Doctor | Hi |
| *Sander* | *Hello* |
| Doctor | How can I help you? |
| *Sander* | *I have a back pain.* |
| Doctor | When did this first occur? |
| *Sander* | *When I lifted a heavy object.* |
| Doctor | Oh, yes then you need some pain killers for this. |
| *Sander* | *Thank you doctor* |

Table 6: Sander in conversation with the doctor

Sander gives the doctor the information he needs to come to the conclusion he himself probably already had drawn. We could even expect him to ask for the medicine immediately, however since this could make the doctor question his motives (he could be addicted for example) he decides not to do this. The doctor however doesn't go into the source of the problem. He just assumed the patient overstretched himself because he lifted something heavy.

### 7.1.2 Susie the ENFP

As an ENFP, Susie has the dominant function of extroverted intuition $N_e$ and as auxiliary function of introverted feeling $F_i$. Therefore these functions should be most dominant in the dialogue. $N_e$ focuses on finding possibilities in situations and $F_i$ is a internal value based judgement function. Combined with each other they make a personality who has strong ideals and is enthusiastic about them. The expected dialogue can be seen in table 7.

| Who | Utterance |
|---|---|
| Doctor | Hi |
| *Susie* | *Hello* |
| *Susie* | *How are you today doctor?* |
| Doctor | I'm good, how can I help you? |
| *Susie* | *I'm afraid I need some medicine* |
| Doctor | Medicine? Why do you need that? |
| *Susie* | *Well, I was watering the plants and all the sudden,* |
| *Susie* | *I got this pain in my back.* |
| *Susie* | *Do you think I'm allergic to plants?* |
| Doctor | Ha ha, no, I think we need to make a scan of your back. |
| Doctor | Because a watering can is a little to light to get back-pain from. |
| *Susie* | *Of course doctor.* |
| Doctor | Can you go to the hospital next Friday at 13:00? |
| *Susie* | *Yes, I will go then.* |

Table 7: Susie in conversation with the doctor

We can now see a stark difference with the INTJ personality. First of all being dominated by extroversion, it was Susie who took the initiative. Secondly she directly asked for medicine, without thinking about the consequences but knowing she probably needs it. Then when explaining the situation she jumped to an idea of why she could have this sudden pain, without thinking about if it even makes sense that you are all the sudden allergic to plants that have been in your home for a while. The doctor however does come to the conclusion that something is odd about getting a back pain from lifting a watering can. So because Susie is more talkative the doctor decides to do more tests rather than just giving some pain killers.

### 7.1.3 Chris the ISTP

With his ISTP type, Chris has the dominant function of $T_i$ and then the auxiliary function of $S_e$. We therefore would expect these functions to do most of the work in the dialogue. $T_i$ uses an internal reasoning structure to make judgments about the world and $S_e$ uses the senses to gather information. The conversation can be seen in table 8.

So this dialogue looks a lot more like that of Sander (INTJ) than that of Susie (ENFP). However the motivation for the responses are quite different than that of Sander. Chris hadn't figured out yet that he needed pain killers when he arrived, since his auxiliary function is $S_e$, he hadn't thought that deep about the problem. He just knew he was in much pain, and knew the doctor could help with that.

| Who | Utterance |
|---|---|
| Doctor | Hi |
| *Chris* | *Hello* |
| Doctor | How can I help? |
| *Chris* | *I have back pain doctor.* |
| Doctor | When did this first occur? |
| *Chris* | *Well I was watering the plants,* |
| *Chris* | *Perhaps I put to much water in the watering can* |
| Doctor | Yes, that could be the case. |
| Doctor | However I would like to make a scan of your back just to be sure. |
| *Chris* | *Can't you just give some pain killers to help me?* |
| Doctor | Yes but that will only work temporary. |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? |
| Doctor | I can give you some pain killers meanwhile. |
| *Chris* | *Okay, thanks doctor* |

Table 8: Chris in conversation with the doctor

The difference with the dialogue of Susie is again quite obvious. He didn't took the initiative because his dominant function isn't extroverted, and unlike Susie he correctly asserted when the doctor asked about it that the object he lifted may have been to heavy.

The conclusion is again different. Because one of the main functions of Chris is $S_e$ he wants to deal with the pain *now*. Therefore he asks the doctor explicitly for pain killers, without considering that only the tests could actually solve the problem permanently. However the doctor comes to a middle ground and besides ordering the test also prescribes painkillers.

### 7.1.4 Influence of personality

So we had 3 different doctor appointments all with the same problem but with different personalities being at play. The end result was three different outcomes for each patient. Sander probably will be back next week with the same complaints at the doctor. However this time his situation may have worsened. Susie will get her problem eventually diagnosed like Chris, however Susie won't have access to painkillers meanwhile. Which may be uncomfortable to her.

From this case study we can conclude that training doctors to deal with different personalities is in fact very desirable because it can allow patients to be treated sooner and more effective. Sander could have had his problem diagnosed a week earlier and Susie could have had access to pain killers for example.

## 7.2 Making a scenario

Making a realistic dialogue can be difficult. Predicting most utterances one actor can make in a particular niche and adding answers to those is even more difficult. This is what AIML asks of its authors. What our modeling system in YAML asks goes a step further, to predict most utterances all actors can make in a particular niche. We will discuss in this section how we modeled

the dialogue and what methods we used to steer particular personalities over particular paths.

### 7.2.1 Dialogue to YAML

The first task is just getting the utterances and the connections (without steering) into our knowledge base (the YAML files). In this section we will discuss how we modeled a part of the scenario discussed in section 7.1. In that section several dialogues are presented with different personalities dealing with the same social practice, visiting the doctor.

Our strategy for modeling these is selecting a particular dialogue and then deciding per utterance whether they: Require a unique symbol, can be merged in an existing one, or already have one and therefore can be ignored. For example we merge the greetings according to listing 29.

```
literals:
  - Hello
  - Hi
```

Listing 29: Greeting modeled

While putting utterances into symbols, we can start thinking about how to group them in scenes. Scenes are available to allow pattern matching to be more general by disabling most patterns at a particular point in the conversation (see section 6.4.3). It's up to the author to decide what is most handy for using this feature. One may even not use this at all and leave everything in the introduction. What we chose to use was the social practice activities in section 2.3 as guide for determining what to group in which scenes.

The final step is modeling the connections. We just look at the dialogue and link up the utterances as can be seen in listing 30. Since we stick to the social practice for organizing the scenes we put the symbol `ask_reason_here` in the `information_gathering` scene (or data gathering in section 2.3).

```
from:
 - greeting
to:
 - symbol: greeting
 - symbol: status
 - symbol: ask_reason_here
   scene: information_gathering
```

Listing 30: Connections in introduction

Although these connections are valid for conversation, the bot has no way to know which personality should choose what path. We have two major ways of steering certain personality kinds over certain paths. Namely, perlocutionary values and goals (see section 4.4). However to use these we first need to setup the initial believes of the bot. Which will be discussed next.

### 7.2.2 Believes

An example `believes.yml` can be seen in listing 31. This file describes several constants in the chatbot, that cannot, or should not be derived. This file in cooperation with the symbols and connection will be used to create an initial believe base.

We need to setup the goals and values, which are the primary steering information sources. The higher a goal is in the goals list the more important it is, with this representation a goal can never be equal in importance. Goals are about informatives being uttered (see section 5.2.1 and section 4.4.2), in our example the bot wants the doctor to utter the symbol `have_painkiller` most, then after that he wants to tell the doctor he has a back pain. Goals are the most powerful method of steering the thinking functions.

We can also see the utility definition of values, higher is more attractive. Values, unlike goals can have an equal utility. Moreover unlike goals their utility can be added upon each other, this does not happen for goals, which brings us to the next difference. Whereas goals are encoded at believes level, values can be encoded in connections themselves (see section 5.2.1 and section 4.4.2). Also note that this is a dictionary definition, meaning it's up to the scenario implementer to decide which value names he wants to use. Finally whereas goals were about steering thinking functions, values are about steering the feeling functions.

Then we need to determine the available actors and which actor is the bot himself. This is required to determine which connections can be used, a connection that does not defines restrictions gets expended into the actor list. There was an idea of trying to derive the available actors from connection definitions, however this wouldn't work in case no actors were ever defined (relying on the expansion mechanism for every connection).

Finally we need to determine the personality this chatbot will use. This is just a sequence of the Jungian function names. There are no restrictions on this, except for your machine resources. You could make a personality of 500 levels deep, but that probably won't run very well.

```yaml
goals:
  - actor: doctor
    scene: diagnoses
    symbol: have_painkillers
  - actor: patient
    scene: information_gathering
    symbol: back_pain
values:
  enthusiasm: 8
  polite: 5

self: patient
actors:
  - patient
  - doctor

# ENFP
personality: [Ne, Fi, Te, Si]
```

Listing 31: Believes YAML file

Loading the believes inside a YAML file presents a design conflict: Should believes be loaded from YAML or inside the Drools? There is a constant pull between these two possibilities. Drools is much more flexible, because it is Turing complete [75], however YAML can be less verbose as just a data format. The issue is that we now indicate that this structure is a standard part of the bot, but it's only part of the personality component. On the other hand one needs to have a method of deciding which connections ought to be used anyway, if you always want to use the same connections for a pattern you may as well

have chosen to use the Alice bot. Doing it this way was also a method of providing a scenario descriptor file, drools however may have been better suited for that. This decision probably needs to be reconsidered once a multilogue architecture will be implemented (see section 5.6).

### 7.2.3 Steering the bot

As discussed before the primary sources of steering the bot are perlocutionary values and goals. However these are not our only tools, if we want to make distinctions between introversion and extroversion we need to look closely at the behavior of the functions in question. For example $T_e$ has as default behavior, meaning no goals present, to prefer connections that transition to another scene, whereas $T_i$ prefers connections that lead to more options (see section 4.4.2).

In listing 32 we can see the extended version of listing 30, with a glance, we can see how the functions maybe influenced by this. For example we expect personalities with a dominant feeling functions to go for the 'status' option, because it gives so much utility. From the status option there are only options that can be uttered by the doctor, this is done simply because it would be strange to continue talking after asking "How are you?".

If the status symbol is liked more by feeling function $F_i$, why did in section 7.3 Susie (with dominant $F_i$) utter a greeting first? The answer is in low level replies (explained in detail in section 5.4), of which we can see an example in listing 33. What this does is execute code upon insertion of a symbol, in this case it just replies with a greeting. So that drool rule will force all personalities to reply a greeting with a greeting. We modelled the connection to add a perlocutionary value to this reply, which means the learning function $F_e$ will prefer uttering polite replies in future choices.

```
1  from:
2   - greeting
3  to:
4   - symbol: greeting
5     values:
6     - Polite
7   - symbol: status
8     restricted_to: patient
9     values:
10    - Polite
11    - Enthusiasm
12  - symbol: ask_reason_here
13    scene: information_gathering
14    restricted_to: doctor
15 ---
16 from:
17  - status
18 to:
19  - symbol: ask_reason_here
20    scene: information_gathering
21    restricted_to: doctor
22    values:
23      - Impatient
24  - symbol: good
25    restricted_to: doctor
26    values:
27      - Polite
28      - Happy
```

Listing 32: Connections in `introduction` extended with values

```
rule "Low level hello replies with hello first time"
when
    $pre:PreProcessed($symbol:utterance.what, $symbol.name == "introduction/greeting", $actor:utterance.byWhom)
    $believes:Believes($actor != self)
then
    log.info("low level entry");
    delete($pre);

    final Informative infor = new Informative($believes.self, $symbol);
    final Utterance resulting = $believes.findToFromLastUttTo(infor)
        .map(Utterance::createFromConnection)
        .orElse(Utterance.create(infor.who, infor.what, PerlocutionaryValueSet.empty));
    insert(new Reply(resulting.setByWhom(infor.who), QueryDatabase.empty));
end
```

Listing 33: Low level greeting reply

If we take listing 34 and the believes into consideration, we can see why thinking functions may like the `ask_reason_here` option: First of all it's a scene transition, which is $T_e$ enjoys, but more importantly, telling about the back pain is a goal in the believes. So any personality that can think ahead with its irrational functions and order with a thinking function would probably like this option.

```
from:
 - ask_reason_here
to:
 - symbol: back_pain
   values:
     - Scary
 - symbol: need_medicine
   values:
     - Concerned
to_defaults:
   restricted_to: patient
```

Listing 34: Connections in `information_gathering`

There are several other tools for steering available too which were discussed in section 6 such as before fields and templates. However these can't be used to steer personality in particular, besides having an influence on the amount of options, and therefore $T_i$. They can also of course be combined with the methods discussed before.

## 7.3 Testing

To verify our implementation's correctness we use several testing methodologies. First at the lowest level we use the so called unit tests to verify individual Jungian functions work as described in this thesis. We test these individually by creating specific believes and a dialogue tree for input, then we check the output against an expected believe base and dialogue tree.

On the highest level we have the so called validation tests. With these we check if the bot behaves as expected by setting up a scenario, and then executing the bot, after which we run trough the scenario manually.

Traditionally there is also an in-between option called integration tests, which test combined parts of the system, but does so in an automated manner. However we haven't used those because setting them up initially takes much more effort than simple unit tests. Validation tests are quick to setup because the process isn't automated.

### 7.3.1 Validation test

To do validation tests we will try to model the scenario presented as an example in section 7.1. How this is done is discussed in section 7.2. After this we test the bot by playing doctor in the scenario. The results were recorded and can be seen in appendix G.

There were some issues with the scenario itself, however most issues we found were with the functions and their interactions. The first issue real interaction issue which made us reconsider the design was that ENFP didn't utter multiple responses (which was discovered in appendix G.1.2). This was solved as discussed in section 4.5.1, where we reinsert the dialogue tree as drools processing, however with the condition that the functions would prefer actor alternation if equal valuation happens (to prevent infinite loops). The second major issue was the $T_i$ function not using its default behavior (discovered in G.2.3), which lead us to think about the level at which personality functions ought to operate in the dialogue tree in a more structured manner (see section 4.5.2).

# 8 In conclusion

In this work we have shown a possible interpretation of Carl Jung's theory of types in a chatbot. We built on top of the work of an existing chatbot, but decided to replace the core modeling language with an alternative because there was no space to do deliberation in. This effort resulted in a more flexible, less verbose and more precise method of modelling dialogue. It also made Drools the center of the new chatbot, where each step of processing a user message was decided by business rules.

Once this was done we could implement our interpretation of Jung and set up a test scenario to test the ideas presented in section 4.4. From the test results in appendix G.8, we can see that the theory works in the implemented chatbot and scenario for the INTJ, ENFP and ISTP personalities. This could in future work help doctors train in dealing with various personalities as discussed in the introduction. We did not prove however that this will work in all instances. A first step towards that would be to model the theories in section 4 in a formal language discussed in section 2.2.3. We haven't had a need to do such an effort yet, and were more concerned with getting a working implementation.

## 8.1 Discussion

There are some odds and ends that haven't been addressed. A really major one is the way we expected the function attitude order type to be and how it was. Section 4.4 was expecting an alternating order of rational and irrational functions, so $ABAB$. However upon closer inspection of MBTI specific theory it, turned out to be a rotating order, along the lines of $ABBA$. Although this has little effect upon the dominant and auxiliary functions, this may be problematic for the deeper functions. Additional experimentation is required to verify this problem, and potentially start looking for a solution.

Another issue is that although the bot is functional, the way we implemented the rational functions is rather simplistic. No actual mini-max algorithm is used (or something similar that would work with non zero sum games). We instead just use static evaluation at each level. So not all dialogue tree information is used in the deliberation, addressing this issue could lead to more varied personalities. Alternative implementations could also be developed for the irrational functions, we leave speculation about these as an exercise to the reader.

## 8.2 Future work

Considering the move to position Drools in the center of chatbot deliberation, and a novel alternate take on modeling chatbot dialogue, there are a legion of possibilities to extend this work. In this section we will present a brief overview of the more obvious extensions, but we expect there will be many niches we haven't considered.

### 8.2.1 Social practice support

We already hinted in section 2.3 that social practices were a consideration. However during the implementation we haven't looked deep into this, except for the fact that Drools maybe a good bases to do this. Although it was already

in place, we just made it the center of deliberation. This will make modelling norms for example more easy, for example we can consider it to be not normal when someone utters the same symbol over and over. Since utterances are already recorded into believes for some of the Jungian functions, we can define a norm with a Drools query. Then we can react by moving to a particularly designed scene for repeating one self. There is much more that this can do, which is discussed thoroughly in section 2.3.

### 8.2.2 Multilogue implementation

We discussed the architectural changes required for multilogue in section 5.6. However as discussed in section 5.6.3, there are some deeper difficulties with doing this, the biggest one is timing and interruption. Scheduling is hard (although well studied [138]). Timing is hard, considering there is little research on for example chatbot initiative, but this paper [139] considered bot-bot interaction trough AIML. Since AIML was used in that paper we can safely say they weren't planning on making a multilogue architecture, because AIML can't do that as discussed in section 6.4.2.

As discussed in section 3.1.1, there already exists a system that can do this, but it's drastically different from this architecture. It may be worth more study if one were to take an endeavor to a multilogue based system.

### 8.2.3 Better use of linguistic theory

Currently linguistic theory is only used on a very basic level, in case of perlocutionary values for example. If symbols and templates were marked with slot grammar [140], a more precise use of templates could happen. So connections that go to a symbol that require a noun, would no longer be able to use symbols that matched a verb.

With help of the work of convolution kernels for semantic parsing [104], a simple knowledge base could be constructed as an inspiration for bot about what to fill into the template system. So that bot to bot conversations could also use the symbols with templates. Care must be taken in this case that the connections are still aligned properly, and the insertion of a template does not alter the meaning of a symbol completely. Although currently this is also a potential problem.

### 8.2.4 More advanced learning

The idea of dynamically extending the symbol graph, and creating new symbols themselves is an interesting one. For example a new symbol is created every time the bot doesn't know something, then the bot asks about it and if the user replies with something which is known, connect them. Currently there is limited support for learning in the $S_i$ function. but this is based on knowing existing symbols, the extension would add symbols dynamically. This could be done by more advanced language parsing techniques in combination with an existing knowledge base such as OWL [105].

### 8.2.5 Matching

It should be noted that regular expressions is just one of the possible ways of determining what symbol the user uttered. The bot can be easily modified to use alternate matching mechanisms such as trough a recurrent neural network for example [98]. What needs to be done is just replace the existing pattern matching drool rule with an alternative and modify the recurrent neural network to select symbols. With probability theory matching could also be made such as done in [141], this would require slight modification of the ideas proposed in that paper. But the core problem is the same: Figuring out what was said, based upon a set of possibilities. We have symbols in a scene whereas they had a tag set. Finally there could be chosen to improve the existing regular expression approach, for example by executing it in parallel, or even upon the GPU [142].

The question of what to do when multiple symbols are matched is a difficult one. In the initial approach we simply selected the first symbol, however currently they are all selected, and we hope the scenario is designed in a way to deal with that. What should be looked into is if there are ways to combine symbols, or perhaps even select one or several based upon personality. Another approach would be to try and use information theory to select the most precise symbol. However this precision could also be encoded or derived somehow (for example, more words means more information).

Note that at the matching phase personality could play a role, for example preferring to match on different symbols, or if multiple are matched, having a preference for certain kinds of symbols. Sensing types may for example prefer raw data based symbols, whereas intuition types would prefer possibilities and ideas.

### 8.2.6 Emotions

In section 2.2.2 the Fatima architecture is discussed that uses decay rates for emotions. Analyzing this could be a good start for re-adding emotions. They used to be a part of the system, however they were based upon type insertions in the AIML files, and couldn't influence dialogue beyond changing scenes. Now the dialogue is much more dynamic because it's completely loaded into memory. However to re-add this, a theory has to be devised of combing OCC with Jung's functions.

If we use perlocutionary values as prime source for emotion tracking, then the model can be extended with information on what expected effect the a perlocutionary value has on the bot. Once an utterance is made, the emotional effects are then applied and decay rates are used to slowly return to the norm. With this new model, the bot could predict which connections could change his emotion to a more desirable state. Personality could play a role in the way it would make these predictions. In that way we only would need to encode the expected emotional change per perlocutionary value, and which emotions are desirable to be in. Personality could also play a role in the desire to be in a particular state, for example a thinking dominated personality would rather be calm, whereas an intuitive personality would rather be excited.

### 8.2.7 Graphical scenario editor

A final possible extension would be to create a graphical scenario editor. Doing this always costs more time than expected and probably should be avoided until above mentioned systems are in place.

# 9 Acknowledgements

There it is, my drop of contribution into the ocean of science. I did this work of course not on my own. Many thanks to my teacher Dr. F. Dignum, for helping me ask the right questions, keeping me focused, and be critical in general about the thesis. Also thanks to Dr. M. Gentile for helping with implementation matters. Finally I want to tank my parents for providing me with housing and a (mostly) calm working environment.

# A References


[1] Andre Campos, Frank Dignum, and Virginia Dignum. Modelling agent reasoning according to a pesonality type. Technical report, 2009.

[2] William Swartout, Ron Artstein, Eric Forbell, Susan Foutz, H Chad Lane, Belinda Lange, Jacquelyn Ford Morie, Albert Skip Rizzo, and David Traum. Virtual humans for learning. *AI Magazine*, 34(4):13–30, 2013.

[3] Johan Jeuring, Frans Grosfeld, Bastiaan Heeren, Michiel Hulsbergen, Richta IJntema, Vincent Jonker, Nicole Mastenbroek, Maarten Smagt, Frank Wijmans, Majanne Wolters, et al. Demo: Communicate!—a serious game for communication skills. *Technical Report Series*, (UU-CS-2015-009), 2015.

[4] Agnese Augello, Manuel Gentile, and Frank Dignum. Social agents for learning in virtual environments. In *Games and Learning Alliance*, pages 133–143. Springer, 2016.

[5] Gillian B Clack, Judy Allen, Derek Cooper, and John O Head. Personality differences between doctors and their patients: implications for the teaching of communication skills. *Medical education*, 38(2):177–186, 2004.

[6] Ref. 143, p. 3.

[7] Ref. 143, p. 4.

[8] Lawrence A Pervin, Oliver P John, and Richard W Robins. *Handbook of personality: Theory and research*. The Guilford Press, 2008.

[9] Lewis R Goldberg. Language and individual differences: The search for universals in personality lexicons. *Review of personality and social psychology*, 2(1):141–165, 1981.

[10] Ernest C Tupes and Raymond E Christal. Recurrent personality factors based on trait ratings. Technical report, DTIC Document, 1961.

[11] Varimax rotation - wikipedia. `https://en.wikipedia.org/wiki/Varimax_rotation`, 2016. Accessed: 2016-12-01.

[12] Leandre R Fabrigar, Duane T Wegener, Robert C MacCallum, and Erin J Strahan. Evaluating the use of exploratory factor analysis in psychological research. *Psychological methods*, 4(3):272, 1999.

[13] Raymond B Cattell. Confirmation and clarification of primary personality factors. *Psychometrika*, 12(3):197–220, 1947.

[14] Jeanne H Block and Jeanne Block. The role of ego-control and ego-resiliency in the organization of behavior. In *Development of cognition, affect, and social relations: The Minnesota symposia on child psychology*, volume 13, pages 39–101, 1980.

[15] HG Gough and P Bradley. California psychology inventory administrator's guide, 1987.

[16] Ref. 8, p. 114.

[17] Ref. 8, p. 114..116.

[18] Ref. 8, p. 119.

[19] Paul T Costa and Robert R MacCrae. *Revised NEO personality inventory (NEO PI-R) and NEO five-factor inventory (NEO FFI): Professional manual*. Psychological Assessment Resources, 1992.

[20] Hans J Eysenck. Four ways five factors are not basic. *Personality and individual differences*, 13(6):667–673, 1992.

[21] Jan Allbeck and Norman Badler. Toward representing agent behaviors modified by personality and emotion. *Embodied Conversational Agents at AAMAS*, 2:15–19, 2002.

[22] Funda Durupinar, Jan Allbeck, Nuria Pelechano, and Norman Badler. Creating crowd variation with the ocean personality model. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1217–1220. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[23] Marlon E. Etheredge. Personality in argumentative agents. `http://dspace.library.uu.nl/handle/1874/341266`, 2016.

[24] Ref. 144, p. 97-98.

[25] Mark R Beauchamp, Alan Maclachlan, and Andrew M Lothian. Communication within sport teams: Jungian preferences and group dynamics. *The Sport Psychologist*, 19:203–220, 2005.

[26] Ref. 144, p. 98-100.

[27] Ref. 144, p. 100-101.

[28] Introverted feeling (fi) explained - one of your eight cognitive functions. `https://www.careerplanner.com/8CognitiveFunctions/Introverted-Feeling.cfm`, 2017. Accessed: 2017-01-07.

[29] Extraverted sensing (se) explained - one of your eight cognitive functions. `https://www.careerplanner.com/8CognitiveFunctions/Extraverted-Sensing.cfm`, 2017. Accessed: 2017-01-07.

[30] Extraverted feeling (si) explained, one of your eight cognitive functions. `https://www.careerplanner.com/8CognitiveFunctions/Introverted-Sensing.cfm`, 2017. Accessed: 2017-01-07.

[31] Extraverted intuition (ne). `http://personalitygrowth.com/extraverted-intuition/`, 2017. Accessed: 2017-01-07.

[32] Ref. 144, p. 104.

[33] Ref. 144, p. 105.

[34] Ref. 144, p. 23.

[35] John G Carlson. Recent assessments of the myers-briggs type indicator. *Journal of Personality Assessment*, 49(4):356–365, 1985.

[36] Mary H McCaulley. Myers-briggs type indicator: A bridge between counseling and consulting. *Consulting Psychology Journal: Practice and Research*, 52(2):117, 2000.

[37] The myers & briggs foundation – understanding mbti type dynamics. http://www.myersbriggs.org/my-mbti-personality-type/understanding-mbti-type-dynamics/, 2016. Accessed: 2016-11-24.

[38] If you're confused about your myers-briggs personality type, read this: An intro to cognitive functions | thought catalog. http://thoughtcatalog.com/heidi-priebe/2015/06/if-youre-confused-about-your-myers-briggs-personality-type-read-this-an-intro-to-cog 2016. Accessed: 2016-11-24.

[39] David Keirsey. *Please understand me 2*. Prometheus Nemesis Book Company, 1998.

[40] Why are the mbti types grouped as sj, nf, nt, and sp when some groups don't share any cognitive functions in common? : mbti. https://www.reddit.com/r/mbti/comments/2dig8v/why_are_the_mbti_types_grouped_as_sj_nf_nt_and_sp/, 2016. Accessed: 2016-11-16.

[41] John Sample. The myers-briggs type indicator and od: implication for practice from research. *Organization Development Journal*, 22(1):67, 2004.

[42] Tammy L Bess and Robert J Harvey. Bimodal score distributions and the myers-briggs type indicator: fact or artifact? *Journal of personality assessment*, 78(1):176–186, 2002.

[43] Daniel W Salter, Deanna S Forney, and Nancy J Evans. Two approaches to examining the stability of myers-briggs type indicator scores. *Measurement and Evaluation in Counseling and Development*, 37(4):208, 2005.

[44] Randolph C Arnau, Bradley A Green, David H Rosen, David H Gleaves, and Janet G Melancon. Are jungian preferences really categorical?: an empirical investigation using taxometric analysis. *Personality and Individual Differences*, 34(2):233–251, 2003.

[45] Gary J Sipps, Ralph A Alexander, and Larry Friedt. Item analysis of the myers-briggs type indicator. *Educational and Psychological Measurement*, 45(4):789–796, 1985.

[46] David J Pittenger. Measuring the mbti... and coming up short. *Journal of Career Planning and Employment*, 54(1):48–52, 1993.

[47] Ref. 144, p. 106.

[48] Frederick Kier and Bruce Thompson. A new measure of jungian psychological types for use in counseling. 1997.

[49] Jason E King, Janet G Melancon, and Bruce Thompson. Score validation and theory elaboration of a jungian personality measure. 1999.

[50] Janet G Melancon and Bruce Thompson. Measurement of self-perceptions of jungian psychological types. 1996.

[51] Bruce Thompson and Elizabeth Stone. Concurrent validity of scores from an adjectival self-description checklist in relation to myers-briggs type indicator (mbti) scores. 1994.

[52] Michael S Matell and Jacob Jacoby. Is there an optimal number of alternatives for likert scale items? study. *Educational and psychological measurement*, 31:657–674, 1971.

[53] Ron Garland. The mid-point on a rating scale: Is it desirable. *Marketing bulletin*, 2(1):66–70, 1991.

[54] Randolph C Arnau, Bruce Thompson, and David H Rosen. Alternative measures of jungian personality constructs. *Measurement and evaluation in counseling and development*, 32(2):90, 1999.

[55] Randolph C Arnau, David H Rosen, and Bruce Thompson. Reliability and validity of scores from the singer-loomis type deployment inventory. *Journal of Analytical Psychology*, 45(3):409–426, 2000.

[56] Mary Loomis and June Singer. Testing the bipolar assumption in jung's typology. *Journal of Analytical Psychology*, 25(4):351–356, 1980.

[57] Adrian Furnham. The big five versus the big four: the relationship between the myers-briggs type indicator (mbti) and neo-pi five factor model of personality. *Personality and Individual Differences*, 21(2):303–307, 1996.

[58] Randolph C Arnau, Bruce Thompson, and David H Rosen. Measurement of jungian personality types. 1997.

[59] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[60] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(02):115–152, 1995.

[61] Michael Bratman. Intention, plans, and practical reason. 1987.

[62] Anand S Rao, Michael P Georgeff, et al. Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995.

[63] Nicholas R Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[64] Ruth Aylett, Brigitte Krenn, Catherine Pelachaud, and Hiroshi Shimodaira. *Intelligent virtual agents*. Springer, 2013.

[65] João Dias and Ana Paiva. Feeling and reasoning: A computational model for emotional characters. In *Portuguese Conference on Artificial Intelligence*, pages 127–140. Springer, 2005.

[66] 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.

[67] Mei Lim, João Dias, Ruth Aylett, and Ana Paiva. Improving adaptiveness in autonomous characters. In *Intelligent virtual agents*, pages 348–355. Springer, 2008.

[68] Philip R Cohen and Hector J Levesque. Intention is choice with commitment. *Artificial intelligence*, 42(2-3):213–261, 1990.

[69] John-Jules Ch Meyer et al. Logics for intelligent agents and multi-agent systems. In *Handbook of the History of Logic*, pages 629–658. Elsevier, 2014.

[70] Anand S Rao and Michael P Georgeff. Modeling rational agents within a bdi-architecture. *KR*, 91:473–484, 1991.

[71] Lars Braubach, Winfried Lamersdorf, and Alexander Pokahr. Jadex: Implementing a bdi-infrastructure for jade agents. 2003.

[72] Mehdi Dastani. 2apl: a practical agent programming language. *Autonomous agents and multi-agent systems*, 16(3):214–248, 2008.

[73] Drools documentation. `https://docs.jboss.org/drools/release/6.5.0.Final/drools-docs/html_single/#d0e4344`, 2016. Accessed: 2017-05-26.

[74] Bc Matej Čimbora. *Usability improvements of OptaPlanner Benchmarker*. PhD thesis, Masarykova univerzita, Fakulta informatiky, 2015.

[75] DVI Weppenaar and HJ Vermaak. Solving planning problems with drools planner a tutorial. *Interim: Interdisciplinary Journal*, 10(1):91–109, 2011.

[76] Piotr J Gmytrasiewicz and Christine L Lisetti. Emotions and personality in agent design. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 360–361. ACM, 2002.

[77] Anne MP Canuto, André MC Campos, João Carlos Alchiere, Eliane CM de Moura, Araken M Santos, Emamuel B dos Santos, and Rodrigo G Soares. A personality-based model of agents for representing individuals in working organizations. In *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 65–71. IEEE, 2005.

[78] Tim Sheard. A pure language with default strict evaluation order and explicit laziness. In *Haskell Workshop*. Citeseer, 2003.

[79] John Launchbury. A natural semantics for lazy evaluation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 144–154. ACM, 1993.

[80] Andreas Reckwitz. Toward a theory of social practices: A development in culturalist theorizing. *European journal of social theory*, 5(2):243–263, 2002.

[81] Ana Luiza B Smolka. Social practice and social change: Activity theory in perspective. *Human Development*, 44(6):362–367, 2001.

[82] Ref. 145, p. 3.

[83] Ref. 145, p. 10.

[84] Elizabeth Shove and Mika Pantzar. Consumers, producers and practices understanding the invention and reinvention of nordic walking. *Journal of consumer culture*, 5(1):43–64, 2005.

[85] Georg Holtz. Generating social practices. *Journal of Artificial Societies and Social Simulation*, 17(1):17, 2014.

[86] Virginia Dignum and Frank Dignum. Contextualized planning using social practices. In *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pages 36–52. Springer, 2014.

[87] Agnese Augello, Manuel Gentile, and Frank Dignum. Social practices for social driven conversations in serious games. In *International Conference on Games and Learning Alliance*, pages 100–110. Springer, 2015.

[88] Agnese Augello, Manuel Gentile, Lucas Weideveld, and Frank Dignum. A model of a social chatbot. In *Intelligent Interactive Multimedia Systems and Services 2016*, pages 637–647. Springer, 2016.

[89] Ref. 146, p. 241–245.

[90] Grice's maxims. `https://www.sas.upenn.edu/~haroldfs/dravling/grice.html`, 2017. Accessed: 2017-01-19.

[91] Richard S Wallace. Don't read me-alice and aiml documentation. *Online at http://www. alicebot. com/dont. html*, 2001.

[92] Adjamir M Galvao, Flavia A Barros, Andre MM Neves, and Geber L Ramalho. Persona-aiml: An architecture developing chatterbots with personality. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1266–1267. IEEE Computer Society, 2004.

[93] David R Traum and Staffan Larsson. The information state approach to dialogue management. In *Current and new directions in discourse and dialogue*, pages 325–353. Springer, 2003.

[94] Joachim Walter. Approaches to dialogue system development: Trindikit vs. cslu toolkit.

[95] Wayne Wobcke, Van Ho, Anh Nguyen, and Alfred Krzywicki. A bdi agent architecture for dialogue modelling and coordination in a smart personal assistant. In *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 323–329. IEEE, 2005.

[96] Eric Bilange. A task independent oral dialogue model. In *Proceedings of the fifth conference on European chapter of the Association for Computational Linguistics*, pages 83–88. Association for Computational Linguistics, 1991.

[97] Bayan Abu Shawar and Eric Atwell. Chatbots: are they really useful? In *LDV Forum*, volume 22, pages 29–49, 2007.

[98] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[99] MPJ Penning. Boosting luck: Improving the language understanding capabilities of kaitito. Master's thesis, University of Twente, 2007.

[100] Karel Van den Bosch, Arjen Brandenburgh, Tijmen Joppe Muller, and Annerieke Heuvelink. Characters with personality! In *International Conference on Intelligent Virtual Agents*, pages 426–439. Springer, 2012.

[101] Adjamir M Galvão, Flávia A Barros, André MM Neves, and Geber L Ramalho. Adding personality to chatterbots using the persona-aiml architecture. In *Ibero-American Conference on Artificial Intelligence*, pages 963–973. Springer, 2004.

[102] Rich Hickey. Rails conf 2012 keynote: Simplicity matters. `https://www.youtube.com/watch?v=rI8tNMsozo0&t=67s`, 2012. Accessed: 2017-06-06.

[103] Jung on the transcendent function - jungian center for the spiritual sciences. `http://jungiancenter.org/jung-on-the-transcendent-function/#_ftn7`, 2016. Accessed: 2016-12-07.

[104] Alessandro Moschitti. A study on convolution kernels for shallow semantic parsing. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 335. Association for Computational Linguistics, 2004.

[105] World Wide Web Consortium et al. Owl 2 web ontology language document overview. 2012.

[106] Simon Peyton Jones. *Haskell 98 language and libraries: the revised report*. Cambridge University Press, 2003.

[107] Partial application - haskellwiki. `https://wiki.haskell.org/Partial_application`, 2016. Accessed: 2016-12-09.

[108] George W Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

[109] Re: Licensing and the library version of git [lwn.net]. `https://lwn.net/Articles/193245/`, 2006. Accessed: 2017-05-05.

[110] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[111] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[112] Dave Thomas and A Hunt. Orthogonality and the dry principle, 2003.

[113] language design - are null references really a bad thing? - software engineering stack exchange. https://softwareengineering.stackexchange.com/questions/12777/are-null-references-really-a-bad-thing, 2010. Accessed: 2017-05-13.

[114] Yoav Zibin, Alex Potanin, Paley Li, Mahmood Ali, and Michael D Ernst. Ownership and immutability in generic java. In *ACM Sigplan Notices*, volume 45, pages 598–617. ACM, 2010.

[115] New activity diagram beta syntax and features. http://plantuml.com/activity-diagram-beta#swimlane, 2017. Accessed: 2017-05-19.

[116] Ralf Lämmel and Simon Peyton Jones. *Scrap your boilerplate: a practical design pattern for generic programming*, volume 38. ACM, 2003.

[117] David Klotz, Johannes Wienke, Julia Peltason, Britta Wrede, Sebastian Wrede, Vasil Khalidov, and Jean-Marc Odobez. Engagement-based multi-party dialog with a humanoid robot. In *Proceedings of the SIGDIAL 2011 Conference*, pages 341–343. Association for Computational Linguistics, 2011.

[118] Simon Keizer, Mary Ellen Foster, Oliver Lemon, Andre Gaschler, and Manuel Giuliani. Training and evaluation of an mdp model for social multi-user human-robot interaction. In *Proceedings of SIGDIAL*, 2013.

[119] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[120] Aiml 1.0.1 (a.l.i.c.e. ai foundation). http://www.alicebot.org/TR/2011/, 2011. Accessed: 2017-04-20.

[121] Json. http://www.json.org/, 2017.

[122] Yaml ain't markup language (yaml™) version 1.2. http://www.yaml.org/spec/1.2/spec.html, 2009. Accessed: 2017-04-16.

[123] Extensible markup language (xml) 1.0 (fifth edition). https://www.w3.org/TR/REC-xml/, 2008. Accessed: 2017-04-16.

[124] toml/toml-v0.4.0.md at master · toml-lang/toml. https://github.com/toml-lang/toml/blob/master/versions/en/toml-v0.4.0.md, 2015. Accessed: 2017-04-22.

[125] Yaml compared to xml. https://stackoverflow.com/questions/1308536/yaml-compared-to-xml, 2009. Accessed: 2017-04-16.

[126] James H Coombs, Allen H Renear, and Steven J DeRose. Markup systems and the future of scholarly text processing. *Communications of the ACM*, 30(11):933–947, 1987.

[127] Json as configuration. https://arp242.net/weblog/json_as_configuration_files-_please_dont, 2016. Accessed: 2017-04-22.

[128] toml-lang/toml: Tom's obvious minimal langauge. https://github.com/toml-lang/toml, 2017. Accessed: 2017-04-22.

[129] Yaml: probably not so great after all. https://arp242.net/weblog/yaml_probably_not_so_great_after_all.html, 2016. Accessed: 2017-04-22.

[130] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[131] Matcher (java platfrom se 8. https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html, 2016. Accessed: 2017-05-19.

[132] The grails framework 3.1.1. http://docs.grails.org/3.1.1/ref/DomainClasses/constraints.html, 2017. Accessed: 2017-05-24.

[133] Matcher (java platfrom se 8. https://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html#group-int-, 2016. Accessed: 2017-04-28.

[134] The apache velocity project. https://velocity.apache.org/, 2016. Accessed: 2017-04-28.

[135] Freemarker java template engine. http://freemarker.org/, 2017. Accessed: 2017-04-28.

[136] Stringtemplate. http://www.stringtemplate.org/, 2017. Accessed: 2017-04-28.

[137] Terence John Parr. Enforcing strict model-view separation in template engines. In *Proceedings of the 13th international conference on World Wide Web*, pages 224–233. ACM, 2004.

[138] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[139] Ulrike Spierling, Sebastian A Weiß, and Wolfgang Müller. Towards accessible authoring tools for interactive storytelling. In *International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pages 169–180. Springer, 2006.

[140] Michael C McCord. Slot grammar. In *Natural language and logic*, pages 118–145. Springer, 1990.

[141] Wei Jin, Hung Hay Ho, and Rohini K Srihari. Opinionminer: a novel machine learning system for web opinion mining and extraction. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1195–1204. ACM, 2009.

[142] Yuan Zu, Ming Yang, Zhonghu Xu, Lin Wang, Xin Tian, Kunyang Peng, and Qunfeng Dong. Gpu-based nfa implementation for memory efficient high speed regular expression matching. In *ACM SIGPLAN Notices*, volume 47, pages 129–140. ACM, 2012.

[143] Walter Mischel, Yuichi Shoda, and Ozlem Ayduk. Introduction to toward an integrative science of the person, hoboken, 2008.

[144] Calvin S Hall and Vernon J Nordby. *A primer of Jungian psychology.* Taplinger, 1973.

[145] Yrjo Engestrom, Reijo Miettinen, and Raija-Leena Punamaki. *Perspectives on activity theory.* Cambridge University Press, 1999.

[146] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2008.

# B List of figures

# C List of tables

# D Symbol overview

| Symbol | Constraints | Description | |
|---|---|---|---|
| $f_a$ | | Function attitudes | |
| $\mathcal{B}$ | | Set of all possible believes | |
| $t$ | | time | |
| $B$ | $B \subseteq \mathcal{B}$ | Believe $B$ | |
| $\Pi$ | | Set of all possible sense information | [8] |
| $\pi$ | $\pi \subseteq \Pi$ | Perception information $\pi$ | [8] |
| $\mathcal{D}$ | | Set of all possible actions | [8] |
| $\Delta$ | $\Delta \subseteq \mathcal{D}$ | Set of actions $\Delta$ executed | [8] |
| $\sigma$ | | A string | |
| $\mathcal{S}$ | | All encoded symbols | |
| $s$ | $s \in \mathcal{S}, s = (\{\sigma\}, \sigma)$ | A symbol $s$ | |
| $g$ | $\sigma \xrightarrow{g} s$ | Mapping function from $\sigma$ to $s$ | |
| $g'$ | $s \xrightarrow{g'} \sigma$ | Mapping function from $s$ to $\sigma$ | |
| $\mathcal{P}$ | | Set of all encoded perlocutionary speech acts | |
| $P$ | $P \subseteq \mathcal{P}$ | Set of perlocutionary speech acts | |
| $p$ | $p \in \mathcal{P}$ | A perlocutionary speech act value | |
| $\Lambda$ | | Set of all active actors | |
| $a$ | $a \in \Lambda$ | an actor | |
| $u$ | $u = (P, a, s, t)$ | Utterance made by $a$ | |
| $D$ | $D = (u, [D])$ | Dialogue tree | |
| $G$ | | A set of connections | |
| $c$ | $c = (P, A, s_1, s_2), c \in G$ $s_1, s_2 \in \mathcal{S} \wedge s_1 \neq s_2$ | Connection, from one symbol to another | |
| $i$ | | An integer | |
| $h$ | $p \xrightarrow{h} i$ | Mapping function from $p$ to $i$ | |
| $\Phi$ | | Set of goals in an agents believe base | |
| $\phi$ | $\phi \in \Phi, \phi = (a, s)$ | A single goal, consisting of actor and symbol | |

Table 9: Symbol table

[8] Not used in implementation

# E Source

A free variant[9] of the software is available at `https://jappieklooster.nl/chatbot`. Note that this source is **not** the same as the salve game therefore the build instructions in appendix F don't work for this source. We published the parts which we had copyright over and rewrote some parts we didn't have copyright over such as the server and client. In the process we also made building a lot easier.

Building happens with Gradle, which provides a script[10] to install all dependencies (including Gradle):

```
1  ./graldew run
```

[9] LGPL-2.1 license for the chatbot core and MIT license for reference implementation

[10] You do require a Java SDK to execute this script, for example: `http://openjdk.java.net/install/index.html`

# F Building salve

To build the salve game two hurdles need to be overcome, because the server uses a starkly different tool chain than the client. In this appendix we will record how the application can be build. It may seem trivial but the Java EE world is incredibly complex. We assume a UNIX-like operating system with a package manager.

Note that these instructions are **not** for the source described in appendix E

## F.1 Client

The client is relatively easy too setup since it's build with a monolithic environment. You need to have the unity editor. The only issues with the client were an incomplete merge and a dangling import that produced build errors. Also note that there exists a Linux editor, it's just not officially supported (yet) but the latest version can be found here: `https://forum.unity3d.com/threads/unity-on-linux-release-notes-and-known-issues.350256/` Scroll all the way down for the latest release.

Note that the unity client currently doesn't support multiple replies from an agent, because the reply is just inserted in a label, rather than showing a chat history.

## F.2 Server

The server runs on Java, therefore the first step is to install Java. In our case java 8 was used. If your system uses portage you can use the following command:

```
1  # emerge dev-java/icedtea:8
```

### F.2.1 Maven

Then maven needs to be installed since Gradle didn't work:

```
1  # emerge dev-java/maven-bin
```

Maven is the package manager for java software, it downloads and installs dependencies (and dependency dependencies) automatically based on XML configuration. Do note that to use maven you need to setup a `~/.m2/settings.xml` file. I based mine on this: `https://maven.apache.org/settings.html`, with help of: `https://maven.apache.org/ref/3.3.9/maven-settings/settings.html` The active profile should have the name local so that the local profile is used in the maven project (in this case *local*). Otherwise the Wildfly plugin won't deploy the application. To test if maven works go to the `communicate2/communicate/communicate_server` folder and execute:

```
1  $ mvn compile
```

If no errors occur it means the settings are configure right. However we are not done yet since the resulting binary is not executable. It is something called a servlet which is an API for server like applications. To use this binary, we need an application server. Our maven repository and code base has been configured towards *Wildfly*, so we will use that.

**Gradle attempt** it was attempted to replace maven with Gradle, since it's a lot less verbose than maven and easier to setup however doesn't have the `picketlink` extension which Wildfly requires. Therefore Gradle was abandoned and the maven tool was used instead.

### F.2.2 Get Wildfly

Download Wildfly from here: `http://wildfly.org/downloads/`, choose the full web distribution (if you choose the servlet one you'll run into trouble since it doesn't have the datasource subsystem, it took about two days to figure that out). Extract this download somewhere which we will call hence forth $WILDFLY.

### F.2.3 Setup datasource

Now it's time to configure the persistent datasource. The code base can handle sessions, but to deal with user registration and logins and such we need a database. There are two methods, MariaDB and the in ram storage. MariaDB is what the online application uses and it's probably better to stick to that for active development, but if you just want to have a quick look at the server you should use look at section F.2.8.

### F.2.4 MariaDB setup

So first install MariaDB (or MySQL, they are the same, except MariaDB has better defaults):

```
# emerge dev-db/mariadb
```

Then we need to setup the user and database:

```
$ mysql -u root
> create database salve;
> GRANT ALL PRIVILEGES ON salve.* To 'salve'@'localhost'
IDENTIFIED BY 'salve';
```

### F.2.5 MariaDB driver

Now we need to make the application server aware of the database. To do this we first need to install a driver from here: `http://central.maven.org/maven2/mysql/mysql-connector-java/5.1.33/mysql-connector-java-5.1.33.jar` then copy this jar into `$WILDFLY/modules/system/layers/base/com/mysql/driver/main` you probably need to make every folder after base. Also create another file called `module.xml` with the following content:

```
<module xmlns="urn:jboss:module:1.3" name="com.mysql.driver">
 <resources>
  <resource-root path="mysql-connector-java-5.1.33.jar" />
 </resources>
 <dependencies>
  <module name="javax.api"/>
  <module name="javax.transaction.api"/>
 </dependencies>
</module>
```

### F.2.6 Wildfly datasource

Now the *driver* is installed we need to configure it as a datasource. To do this we move to `$WILDFLY/bin`. Then execute the following commands:

```
$ chmod +x add-user.sh jboss-cli.sh standalone.sh
$ ./standalone.sh &
$ ./jboss-cli.sh --connect controller=localhost
--command="/subsystem=datasources/jdbc-driver=mysql:add(driver-name="\
"mysql,driver-module-name=com.mysql.driver,driver-class-name="\
"com.mysql.jdbc.Driver)"
$ ./add-user.sh
$ xdg-open localhost:9990
```

That last command should open the browser. Click then Configuration → subsystems → datasource → non `xa` → add → MySQL → next. The name should be `GameDS` and the `JNDI` name should be `java:/GameDS`, now click: next → detect driver → MySQL. The URL should be `jdbc:mysql://localhost:3306/salve`, the username and pass should both be salve, now click next → finish.

### F.2.7 Deploying

first go to the `communicate2/communicate/communicate_server` folder. Then to deploy the application the following command is used:

```
$ mvn wildfly:deploy
```

If your build gets stuck because it tries to find communicate jars from the internet it can help to go to the project root folder and execute:

```
$ mvn compile
```

### F.2.8 Alternative: in memory db

Note that I didn't test this but it should work. You can either choose to use maria db or try and point the application to the in ram storage of Wildfly. To do this go to: `communicate2/communicate/communicate_server/src/main/resources/META-INF` and then replace everything with:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/xmlschema-instance"
    xsi:schemalocation=
    "http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
    >
        <persistence-unit name="salve_persistence_unit"
            transaction-type="JTA">
            <jta-data-source>java:jboss/myDs</jta-data-source>
            <properties>
                <property name="hibernate.dialect"
                          value="org.hibernate.dialect.H2Dialect" />
                <property name="hibernate.max_fetch_depth" value="3" />
                <property name="hibernate.hbm2ddl.auto" value="update" />
                <property name="hibernate.show_sql" value="true" />
            </properties>
        </persistence-unit>
    </persistence>
```

## F.3 Ubuntu issues

At some point my laptop crashed and I had to fall back to an Ubuntu based distribution. Installing MySQL came with the gotcha, the default root on MySQL is only accessible from:

```
sudo mysql -u root
```

I thought I could do this as the normal user but it appears this not the case. Also an online forum said that you had to run

```
sudo mysql_install_db
```

However I think apt handles that for you.

**utf8 problem** I got an error:

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException:
  Specified key was too long; max key length is 767 bytes
```

This is because it uses an utf8 encoding (see this website: `http://stackoverflow.com/questions/10748155/specified-key-was-too-long-max-key-length-is-767-bytes`). To solve this change the tables to use a smaller utf8 instead. In '/etc/mysql/mariadb.conf.d/50-server.cnf' change the variables to

```
character-set-server  = latin1
collation-server      = latin1_swedish_ci
```

We also need to modify '/etc/mysql/mariadb.conf.d/50-client.cnf' with:

```
default-character-set = latin1
```

After which we do a restart of the `mysqld`:

```
sudo systemctl restart mysql
```

Note: make sure to stop your running Wildfly instance, otherwise Wildfly will keep it alive

Now drop the entire salve database as root, the tables were created with the wrong char-set so we need to just remove them all.

```
mysql -u root -ppassword -e "DROP DATABASE salve;:
```

Now recreate the database as was done in section F.2.4 (alternatively logging in as root and pressing arrow up a couple of times should get you the right commands).

## F.4 Notes

If you're located in the communicate_server folder, The rebuild everything command is:

```
rm -R ~/.m2/repository/cnruithof; (cd ../ && mvn clean && mvn install) && mvn wildfly:deploy
```

There is also a shell script `rebuild.sh`. This will re-install the entire project thereby also rebuilding all dependencies. `mvn clean` in there for good measure.

There is a python client script for quick debugging, therefore it's unnecessary to keep unity running (or use at all).

# G Test Results

All these test follow the scenario which is presented as case study in section section 7.1. Each time all tests are executed to prevent regressions, even if they did manage to pass a test cycle before. Every test we indicate the commit hash on which it was performed upon for reproducibility reasons.

## G.1 Initial test

| | |
|---|---|
| Commit | `65e704a0f9ec6f5d7052e0de285a79baf420db7d` |
| Date | 2017-03-28 |

We believe most of the higher level reasoning is done at this point. Therefore we attempted this acceptance test to see how well the system would hold up.

### G.1.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | 1 |
| Doctor | How can I help you? | 2 |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

1. Actually said "How are you?", This is unexpected, however since replying with hello is useless to INTJ (it'll just loop back), such a reply may actually be better for progress high level process. However, this doesn't mean this is right, the intention was to make the hello reply be done on a lower level (first item said and understanding matches hello short-circuits into replying hello)

2. We use the other scripted response (perhaps it thinks it's ENFP), this didn't match however, turns out this symbol didn't have any regular expressions, after adding the regular expressions it still didn't match, further investigation is required into the regular expressions why they don't match, I'm suspecting it's either a construction problem (not all are added), Or some standard java regex problem.

### G.1.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | 1 |
| *Susie* | *How are you today doctor?* | 1 |
| Doctor | I'm good, how can I help you? | 2 |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | 3 |
| *Susie* | *I got this pain in my back.* | |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | 4 |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

1. Skipped saying hello (again we miss the short circuit rule, same as INTJ)
2. Didn't match with I'm good, probably same issue as INTJ.
3. Only says the first sentence. This is because we don't keep on popping sentences from the dialogue tree if the patient actor still is preferred.
4. We skipped the intermediate sentences (because they didn't pop), What we would expect is to get some confusion back after doing this. This didn't happen so rules to give confusion when skipping parts of the graph should be implemented.

### G.1.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | 1 |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | 2 |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | 3 |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | |
| Doctor | Yes but that will only work temporary. | |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

1. Again no hello
2. Replied with, I'm afraid I need some medicine, I think this is because $T_i$ is the first function, and I think the two pass effect isn't implemented correctly yet. It just goes deeper now rather than finding the right level

to modify. Since telling about back pain is a goal and we have a unit test that ti prefers goals it's almost surely this broken two pass behavior.

3. It says when lifting a heavy object, but I'm pretty sure it just always picks the first option because two pass logic is broken. So I'll end this test prematurely.

## G.2 Second test

| | |
|---|---|
| Commit | `1de05450c187c51df780a975cc4f0cec7c69dba1` |
| Date | 2017-04-05 |

After solving the issues from the initial test, we redid them.

### G.2.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | |
| Doctor | How can I help you? | |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

Conversation went according to script.

### G.2.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | |
| *Susie* | *How are you today doctor?* | 1 |
| Doctor | I'm good, how can I help you? | 2 |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | |
| *Susie* | *I got this pain in my back.* | 3 |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

1. Wasn't uttered, probably because we handle hello as a low level reply. Perhaps we should let replies also be passed to themselves so higher level functions have a chance to interact? (note that alteration is the default, so unless a goal is directly below it nothing will be said)

2. We just said 'how can I help you' instead.

3. Goes into "Perhaps I put to much water in the watering can". Probably because ENFP is an $F_i$ rather than $F_e$, $F_i$ is a learning function [11]. We should analyze what has been learned at this point. It appears also that in either case no values are attached. After this point the bot follows the ISTP script. This is wrong, $F_e$ is the learning function, $F_i$ uses perlocutionary values. So this was easily fixed by modifying the perlocutionary value utility.

### G.2.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | 1 |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | |
| Doctor | Yes but that will only work temporary. | |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

1. Says 'when lifting a heavy object'. This derails the entire script so the test was stopped. The reason for this became apparent after a full tree dump after each function. First time the $T_i$ function has nothing to sort, so it does nothing. $S_e$ generates all available options in the order they came, $N_i$ just goes down whatever $S_e$ preferred. $F_e$ sorts everything twice, but most have no perlocutionary value. Then once we come back to $T_i$, the most obvious choice is of course the first option provided by $S_e$, since it was expended by $N_i$, twice.

This is obviously not what we want, since $S_e$ is deciding which action is taken here. $S_e$ could try and get a sane ordering function. However it only has next available and trying to use previous would lead to a stack overflow. $T_i$ probably should get a better ordering mechanism. For example rather than using the `DialogueTree` for direct options we just list all available options. We ended up doing this but note that this breaks the design idea of the architecture.

To fix this properly we need to reconsider the design at a deeper level, for example give ti the opportunity somehow in with direction $N_e$ will go down. A potential better way of solving this would to allow irrational to either produce on the level of previous irrational, or if the produced action already exists, go one lower. But since we're starting to run into time constraints we just leave it at this 'solution'.

[11] note I got this wrong here in my initial analyses, $F_e$ is the learning function

## G.3 Third test

| | |
|---|---|
| Commit | `893f54f6942a65e3c22126949091d096c3691445` |
| Date | 2017-04-10 |

After solving the issues from the second test we did another run of all tests. Even though this time we tested quite closely against the issues from the previous test, going trough the entire battery is necessary to prevent regressions.

### G.3.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | |
| Doctor | How can I help you? | |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

Conversation went according to script.

### G.3.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | |
| *Susie* | *How are you today doctor?* | |
| Doctor | I'm good, how can I help you? | |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | |
| *Susie* | *I got this pain in my back.* | |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

Conversation went according to script.

### G.3.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | 1 |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | 2 |
| Doctor | Yes but that will only work temporary. | |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | |
| *Chris* | *Yes, I will go then.* | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

1. Said hello twice?! This was caused by the reinserting part of the shortcut, it used any actor rather than self. Fixing this caused a regression in ENFP which wouldn't ask about the doctor's day anymore. This was solved by adding an extra value to that connection (making it more appealing for $F_i$).

2. Said "of course" rather than give me some pain killers After inspection of the scenario it appears that there is no reason to say "can't you give me painkillers" because there weren't any connection leading out of it. I also made give painkillers a low priority goal, to force ISTP in that direction.

## G.4 Fourth test

| | |
|---|---|
| Commit | `8ea894d0da28addd8067c341e842ca2d91296586` |
| Date | 2017-04-10 |

It's the same day and we try another round to try and pass this. Although we modified ISTP slightly, because it had to parse a large amount and produce a single reply for no obvious reason. Unlike ENFP which for example is expected to produce multiple replies because it's talkative.

### G.4.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | |
| Doctor | How can I help you? | |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

Conversation went according to script.

### G.4.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | |
| *Susie* | *How are you today doctor?* | |
| Doctor | I'm good, how can I help you? | |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | |
| *Susie* | *I got this pain in my back.* | |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

Conversation went according to script.

### G.4.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | |
| Doctor | Yes but that will only work temporary. | |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | |
| *Chris* | *Yes, I will go then.* | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

Conversation went according to script. Note however that we split up the utterances of the doctor compared to the scenario. This is because there were to many different symbols in the input (scan and give painkillers), this made the bot reply to both of them.

## G.5 Fifth test

| | |
|---|---|
| Commit | `ae74151591d0c5a20566d8d74283d4c73e9450d2` |
| Date | 2017-04-30 |

This test was done after completing the major features that were still missing in comparison with AIML. We wanted to make sure no regressions had occurred meanwhile.

### G.5.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | |
| Doctor | How can I help you? | |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

Conversation went according to script

### G.5.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | |
| *Susie* | *How are you today doctor?* | |
| Doctor | I'm good, how can I help you? | |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | |
| *Susie* | *I got this pain in my back.* | |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

Conversation went according to script

### G.5.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | |
| Doctor | Yes but that will only work temporary. | 1 |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | 1 |
| *Chris* | *Yes, I will go then.* | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

1. It didn't match these sentences, apparently because of the new scene system. There was no connection for this transition.

## G.6 Sixth test

| | |
|---|---|
| Commit | `e15409ffc37b837a47a9ceb3c183d67f18c3eb17` |
| Date | 2017-04-30 |

Because of the ISTP issues in last test we wanted to try and pass this one.

### G.6.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | |
| Doctor | How can I help you? | |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

Conversation went according to script.

### G.6.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | |
| *Susie* | *How are you today doctor?* | |
| Doctor | I'm good, how can I help you? | |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | |
| *Susie* | *I got this pain in my back.* | |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

Conversation went according to script.

### G.6.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | |
| Doctor | Yes but that will only work temporary. | |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | |
| *Chris* | *Yes, I will go then.* | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

Conversation went according to script.

## G.7 Seventh test

| | |
|---|---|
| Commit | `bda8dcfb77e7f144ae04eb930611ea74f311a8e7` |
| Date | 2017-06-09 |

Before handing in the project I wanted to do a final acceptance test.

### G.7.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | |
| Doctor | How can I help you? | |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

Conversation went according to script.

### G.7.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | |
| *Susie* | *How are you today doctor?* | |
| Doctor | I'm good, how can I help you? | |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | |
| *Susie* | *I got this pain in my back.* | |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

Conversation went according to script.

### G.7.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | |
| Doctor | Yes but that will only work temporary. | |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | |
| *Chris* | *Yes, I will go then.* | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

Conversation went according to script.

## G.8 Eighth test

| | |
|---|---|
| Commit | `fca7c62eb90e8f2021f92f9f74b9468306c1aebb` |
| Date | 2017-06-09 |

Before handing in the project for `rc-2`, I wanted to do another final final acceptance test, this time using the public repository and commit hash (see appendix E).

### G.8.1 INTJ

| Who | Scripted Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Sander* | *Hello* | |
| Doctor | How can I help you? | |
| *Sander* | *I have a back pain.* | |
| Doctor | When did this first occur? | |
| *Sander* | *When I lifted a heavy object.* | |
| Doctor | Oh, yes then you need some pain killers for this. | |
| *Sander* | *Thank you doctor* | |

Conversation went according to script.

### G.8.2 ENFP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Susie* | *Hello* | |
| *Susie* | *How are you today doctor?* | |
| Doctor | I'm good, how can I help you? | |
| *Susie* | *I'm afraid I need some medicine* | |
| Doctor | Medicine? Why do you need that? | |
| *Susie* | *Well, I was watering the plants and all the sudden,* | |
| *Susie* | *I got this pain in my back.* | |
| *Susie* | *Do you think I'm allergic to plants?* | |
| Doctor | Ha ha, no, I think we need to make a scan of your back. | |
| Doctor | Because a watering can is a little to light to get back-pain from. | |
| *Susie* | *Of course doctor.* | |
| Doctor | Can you go to the hospital next Friday at 13:00? | |
| *Susie* | *Yes, I will go then.* | |

Conversation went according to script.

### G.8.3 ISTP

| Who | Utterance | Issues |
|---|---|---|
| Doctor | Hi | |
| *Chris* | *Hello* | |
| Doctor | How can I help? | |
| *Chris* | *I have back pain doctor.* | |
| Doctor | When did this first occur? | |
| *Chris* | *Well I was watering the plants,* | |
| *Chris* | *Perhaps I put to much water in the watering can* | |
| Doctor | Yes, that could be the case. | |
| Doctor | However I would like to make a scan of your back just to be sure. | |
| *Chris* | *Can't you just give some pain killers to help me?* | |
| Doctor | Yes but that will only work temporary. | |
| Doctor | So let's plan a scan at the hospital next Friday at 13:00? | |
| *Chris* | *Yes, I will go then.* | |
| Doctor | I can give you some pain killers meanwhile. | |
| *Chris* | *Okay, thanks doctor* | |

Conversation went according to script.