Towards Automatic Explaining Artificially Intelligent Behaviour

Ramon Hagenaars
Department of Information and Computing Sciences,
Utrecht University,
Utrecht,
the Netherlands,
3508 TB
ramonhagenaars@gmail.com

October 30, 2012

Abstract

This thesis describes the research towards automatically generating textual explanations for the behaviour of techniques of Artificial Intelligence, in order to obtain a greater transparency for HeMAS. HeMAS is an expert-system-like multi agent system with agents that apply divergent Artificial Intelligence techniques. It applies datamining on medical patient data and deduction on the newly formed rules to give advice or a diagnosis concerning individual patients. In this thesis, a solution towards enhanced transparency is proposed that includes the adaption of existing HeMAS agents to make them explainable and the implementation of a new agent that combines all obtained information from the agents and forms coherent explanations. This research is one of the fields of Explainable AI as well as Natural Language Generation and borrows from the field of Argumentation Theory. Involved aspects concerning the generation of explanations are the application of Toulmin Models for the capturing of the reasoning of the agents and the translation of First Order Predicate logic to natural language which is largely covered by LogicBabelfish, a new small programming language for this very purpose. The in this document described solution has been partially implemented and was applied to a small test case. The resulting explanations seem convenient.

Contents

1 Introduction **9**
- 1.1 Motivation . . . 10
  - 1.1.1 HeMAS . . . 10
- 1.2 Research Plan . . . 11
  - 1.2.1 Targets . . . 11
  - 1.2.2 Methodology . . . 12
- 1.3 Related Work . . . 13
  - 1.3.1 MYCIN . . . 13
  - 1.3.2 XPLAIN . . . 14
  - 1.3.3 BDI-based Explainable Agents . . . 14
  - 1.3.4 Shortcomings . . . 15
- 1.4 Outline . . . 15

2 Relevant Literature **17**
- 2.1 Explainable AI . . . 17
  - 2.1.1 The Essence of Explanations . . . 17
  - 2.1.2 Types of Explanations . . . 18
  - 2.1.3 Argumentation Theory for Explaining . . . 19
- 2.2 Natural Language Generation . . . 22
  - 2.2.1 Natural Language Generation Techniques . . . 23
  - 2.2.2 Architecture of Natural Language Generation Systems . . . 23
- 2.3 Chapter Conclusion . . . 24

3 Adequately Explaining HeMAS **27**
- 3.1 Transparency of Configurations . . . 27
- 3.2 Requirements for Explanations . . . 28
  - 3.2.1 Content . . . 28
  - 3.2.2 Presentation . . . 29
- 3.3 Agent Clara . . . 30
- 3.4 Explainable Agents . . . 30
- 3.5 Example Output . . . 31
- 3.6 Chapter Conclusion . . . 31

4 Design **33**
4.1 Overview . . . 33
4.2 Classifying Agents . . . 34
4.2.1 Approaches . . . 35
4.3 A Model for Transitions . . . 36
4.3.1 Applying the Toulmin Model . . . 37
4.3.2 Toulmin Chain . . . 39
4.3.3 Implementing Toulmin Chains . . . 39
4.4 Interlingua for Explanations . . . 40
4.4.1 First Order Predicate Logic . . . 41
4.4.2 Implementing Logic . . . 43
4.5 Translating First Order Predicate Logic to Natural Language . . . 43
4.5.1 Operators . . . 44
4.5.2 Functions . . . 44
4.5.3 Predicates . . . 45
4.5.4 Wordnet as Lexicon . . . 46
4.5.5 General Application . . . 47
4.5.6 External Templates and LogicBabelfish . . . 48
4.6 Example Implementation . . . 49
4.7 Chapter Conclusion . . . 50

5 LogicBabelfish **53**
5.1 Introduction . . . 53
5.1.1 LogicBabelfish Applications . . . 54
5.1.2 LogicBabelfish as NLG Tool . . . 54
5.2 Syntax . . . 55
5.2.1 Definition . . . 55
5.2.2 Reference to External Functions . . . 56
5.2.3 EBNF . . . 57
5.3 Semantics . . . 58
5.3.1 Grounding . . . 59
5.3.2 Definition Calls . . . 61
5.3.3 Cyclic Dependency . . . 62
5.3.4 Priorities . . . 65
5.3.5 Interpretations . . . 66
5.4 General Appliance . . . 66
5.4.1 Default Identifiers . . . 66
5.4.2 Logical Operators . . . 67
5.4.3 Implementing Lexicons . . . 68
5.4.4 Implementing Bridges . . . 69
5.4.5 Surface Realisation . . . 70
5.4.6 A step to Natural Language Understanding . . . 71
5.5 Chapter Conclusion . . . 71

6 Conclusion **73**
6.1 Outline of Contribution 73
6.1.1 Evaluation of the Hypothesis 74
6.2 Future Work 75

Chapter 1

Introduction

Many many millions of years ago a race of hyper-intelligent pan dimensional beings (...) got so fed up with the constant bickering about the meaning of life (...) that they decided to sit down and solve their problems once and for all. At this end they built themselves a stupendous super-computer (...).
...
'O Deep Thought Computer,' he said, 'the task we have designed you to perform is this. We want you to tell us...' he paused, '...the Answer!'
'The Answer?' said Deep Thought. 'The Answer to what?'
'Life!' urged Fook.
'The universe!' said Lunkwill.
'Everything!' they said in chorus.
...
'Tell us!'
'All right,' said Deep Thought. 'The Answer to the Great Question...'
'Yes...!'
'Of Life the Universe and Everything...' said Deep Thought.
'Yes...!'
'Is...' said Deep Thought, and paused.
'Yes...!'
'Is...'
'Yes...!!!...?'
'Forty-two,' said Deep Thought, with infinite majesty and calm.

This short element of the fictional masterpiece, The Hitchhiker's Guide to the Galaxy by Douglas Adams [Adams, 1979], tells about an event in the universe, where intelligent, human-like beings attempt to answer the ultimate questions of life. In order to do so, they built a very advanced sort of expert system, capable of reasoning autonomously. However, the creators are surprised to find that the answer of the most complex issue in the universe is in fact a very simple one; a number of two digits, namely forty-two. Obviously, this was a

very unsatisfying, disappointing and even disturbing answer, which in the end led to the creation of another, even more powerful supercomputer for calculating the ultimate question to which the forty-two should be applied.

Clearly, very short answers like 'yes' or 'no' are seldomly found sufficient by many people. Often, it is not even the answer, but the reasoning towards it that the questioner was actually looking for. For example, the question people frequently asks to friends or colleagues, 'how are you?', is mostly about what the answerer had been doing a past time period leading to his/her current emotional state, rather than just this state alone. As the title of this thesis somewhat previews, the research of which this thesis reports has been conducted to clarifying the behaviour of Artificial Intelligence (AI). It is of importance that the formation of decisions of agents is understandable in order to judge whether the passed reasoning is acceptable or not. The acceptability of the reasoning is inextricably linked to the defensibility of the performed action resulting from this reasoning. This applies particularly to agents that hold an advisory or diagnosing role like expert systems to which such roles are assigned per definition. Advisory and diagnosing systems can only be of use if their advice or diagnose is trusted by its clients. This necessity led to this research to explaining AI behaviour.

This first Chapter introduces the reader to the study that was done of which this thesis is a report. It contains the motivation for the performed research and a short overview of how this research was done. Thereafter a summation of related earlier work is presented and finally, a survey of this thesis is provided.

1.1 Motivation

The described research has been conducted at the Alan Turing Institute Almere, the Netherlands and aims towards a solution for an internal AI system. The following paragraph will introduce the reader to this system.

1.1.1 HeMAS

The Heterogenous Multi Agent System (HeMAS) is a diagnosing, advisory and pharmaceutical system developed to reason about certain medical domains. In this respect, it resembles a medical expert system since it can be queried and returns an advice, diagnose or an applicable drug, combined with a certainty determination to illustrate the system's confidence in its own result. HeMAS consists of multiple heterogeneous components which are called agents. Each agent applies a different AI or pattern recognition technique for inductive or deductive reasoning, from a wide range of state of the art AI techniques such as Boltzmann Machines (neural networks), Naive Bayesian Networks, Regression Analysis and Decision Trees. For input, HeMAS can use datasets of over a thousand entries that are examined by certain (datamining) agents with corresponding AI techniques. These agents discover patterns, extract rules from them and allow other agents to reason about these rules. In this, HeMAS distinguishes itself from most traditional expert systems, since these expert systems

usually apply only one reasoning mechanism about hand-programmed rules. This thesis primarily focusses on the advisory and diagnosing role of HeMAS.

HeMAS can be configured differently, such that the involved agents cooperate, what is believed to improve the reliability of the final outcome. These configurations are designed by a team of AI specialists and medical experts and are dedicated to specific cases, like diseases such as MEN1, obesity and others. This composing of configurations is done in a special visual programming language that is called *Marama*, which is Igbo for 'beautiful'. HeMAS configurations hold the property of recursiveness, meaning that a configuration can be embedded into another configuration such that the former behaves and is treated like any other agent in the latter configuration.
Since HeMAS is intended for practical use in medical domains, it is vital for HeMAS to be transparent enough. As of yet, HeMAS does not provide this transparency sufficiently. This has motivated the research towards automatically explaining AI techniques.

1.2 Research Plan

1.2.1 Targets

As could be derived from the title of this document, the conducted research points at explaining AI placed in HeMAS' context. The research goal follows from the previous Section and is defined as follows:

Research Goal: *To automatically explain the different datamining and AI techniques within HeMAS, such that the ultimate result of the execution of these techniques is more transparent, leading to an improvement of HeMAS' trustworthiness.*

Please note that the transparency of a system is causally related to sufficiently explaining this system. HeMAS is a multi agent system that applies different AI techniques. It therefore is necessary to explain these AI techniques separately. The following two questions follow from this observation and form the focus of this research:

Question 1: *How can the behaviour of an AI technique be sufficiently explained?*

This question addresses how we can explain afterwards what an artificial intelligent system did during its time running. Chapter 3 will mainly address the first research question and attempts to give a possible solution. The second question follows from the first:

Question 2: *How can the automatic generation of explanations be realised in a generic way?*

This question concerns the design of the answer to the first question. It focusses on the development of such a mechanism. Since AI techniques can be quite divergent and since there is no saying what future AI techniques might be invented, answering this question also involves setting conditions to the AI techniques.

From these research questions arises a hypothesis that has been formalised as follows:

Hypothesis: *The design decisions of the solution in this thesis are feasible and lead to an improvement of the trustworthiness of HeMAS.*

It is attempted to answer the hypothesis by means of addressing the research questions. Naturally, the evaluations of the goal, research questions and the hypothesis, can be found in the final Chapter of this document, the conclusion.

1.2.2 Methodology

As is derivable from the research goal stated in the previous Section, the main objective is to come up with a device capable of generating explanations automatically. Clearly, this goal is in fact a design goal, which means that the research reported in this document can be classified as an instance of design science [Simon, 1996]. According to Simon's work, design science or science of the artificial is the science that is concerned with how things ought to be, rather than in natural sciences where is studied how things are.

The conducted research of this thesis can be summarised by the following sequence of events:

1. Determining work that is related to this project and specifying why it cannot be applied such that the research goal is achieved sufficiently
2. Introducing the reader to the literature that is consulted in the progress towards solutions to the research questions
3. Providing an outline (or design) of a sufficient solution
4. Describing the design details of the defined solution
5. Evaluating the solution in relation to the research goal

The described enumeration represents the chronological process of the creation of this thesis. The first event is described in the Section about related work, in the first Chapter. The literature study, the second event, is described in the second Chapter. Event three is covered by the third Chapter, describing how an adequate solution should look. The fourth event concerning the design of the earlier defined solution is described by Chapter four and five. The last event will be addressed by the last Chapter; the conclusion shall contain an evaluation of the solution in context of the research goal.

1.3 Related Work

Automatically explaining software applications or AI systems has been widely studied for the past three decades, particularly within the context of expert systems. This Section covers earlier work that is related to the work that is described in this thesis.

1.3.1 MYCIN

An early expert system that explained its own advice was MYCIN, a computer-based system provided with expert knowledge about infectious disease therapy [Shortliffe and Buchanan, 1975]. This consultation program has been developed in such a way that the consulting physician could recognise his own way of reasoning towards decision making, with the underlying idea that this would improve the trust in MYCIN. All rules were programmed by hand and written in LISP and consisted of premises and actions. On consultation, MYCIN would examine only the relevant rules and take actions where the premises of the rules are believed to be true, or request more input if the information of the premises are insufficient. Figures 1.1 and 1.2 illustrate an example MYCIN rule and its translation, respectively.

```
PREMISE:
            ($AND   (SAME (VAL (CNTXT GRAM) GRAMPOS)
            ($AND   (SAME (VAL (CNTXT MORPH) COCCUS)
            ($AND   (SAME (VAL (CNTXT CONFORM) CHAINS)
            3)
ACTION:     CONCLUDE CNTXT IDENT STREPTOCOCCUS)
            TALLY .7)
```

Figure 1.1: Example MYCIN Rule in Lisp

```
IF:
        THE GRAMSTAIN OF THE ORGANISM IS GRAMPOS, AND
        THE MORPHOLOGY OF THE ORGANISM IS COCCUS, AND
        THE GROWTH CONFORMATION OF THE ORGANISM IS
        CHAINS
THEN:
        CONCLUDE THAT THE IDENTITY OF THE ORGANISM IS
        STREPTOCOCCUS (MODIFIER: THE CERTAINTY TALLY
        FOR THE PREMISE TIMES .7)
```

Figure 1.2: Translation MYCIN Rule in English

1.3.2 XPLAIN

Another expert system comparable to MYCIN was XPLAIN [Swartout, 1983]. Unlike MYCIN, XPLAIN was developed as a research project for the explaining of expert systems and can be seen as an early implementation of *Explainable AI*, the science that targets the explaining of artificial intelligent behaviour. The innovativity of XPLAIN lay in the fact that it not only translated its reasoning trace to English like MYCIN did, it also defended its reasoning process by providing so-called justifying explanations. So to say, it provided answers to the *what* and *why* questions about its trace.

Swartout distinguised three types of questions one may ask an expert system: (1) questions about what the system did to conclude its advice; (2) questions about why certain steps were taken during the reasoning process; (3) questions involving confusion about terminological use. The first type of question can be answered by translating the executed code to English (like MYCIN did). The second question goes deeper and requires a justification of the rules that were used. The third question fell outside XPLAIN's scope. Swartout found that the justifying answer is hard to generate, since it originates from the programmer's mind who programmed the rule and logically, it should be the programmer answering such questions. To solve this problem, XPLAIN was provided with an automatic programmer. Figure 1.3 gives an example of XPLAIN justifying its request for more information.

XPLAIN	:	Is the patient showing signs of paroxysmal atrial tachyeardia with block? (yes or no)
CLIENT	:	Why?
XPLAIN	:	The system is assessing paroxysmal atrial tachycardia with block. If the status of paroxysmal atrial tachycardia with block is equal to present, the assessment of paroxysmal atrial tachycardia with block is set to the assessment level for present findings (1), otherwise the assessment of paroxysmal atrial tachycardia with block is set to the assessment level for absent findings (0)

Figure 1.3: XPLAIN Justifying

1.3.3 BDI-based Explainable Agents

A significantly different approach than MYCIN and XPLAIN is the implementation of explainable agents based on the BDI-model (Beliefs, Desires, Intentions [Bratman, 1987]) as described by [Harbers, 2011]. This BDI-model is a simplification of the relation between human reasoning and acting. BDI agents can have beliefs representing the knowledge[1] about the agent's world, desires stating

[1]Knowledge is a laden term, since philosophically no one can know anything, but believe. 'I know that I know nothing' $\sim$ Plato

the goals that the agent wishes to achieve and intentions, or acting plans for the actual strive after the goals.

Harbers intends the application of explainable agents in a training context. The software agents are virtual colleagues or assistants and the trainee can interact with them in order to improve his social, leading or negotiation abilities. These virtual agents are required to clarify their behaviour when they do not behave as instructed by the trainee. For this reason, Harbers proposes explainable agents that can and do explain their own behaviour.

Harbers defends the use of the BDI-model with the conception that this model is representative for the human cognitive functioning of explanation and thus believes that the agent's explanation should apply these mental concepts as well. Furthermore, the behaviour of intelligent systems is often explained by the line of reasoning towards it and since the BDI-model is believed to suit the human cognitive model for explaining, this reasoning should also adhere to this model. In [Harbers, 2011], six algorithms are presented that can be used to generate explanations for the actions of explainable agents. To summarize, the algorithms describe that the behaviour of agents can be explained by explaining the beliefs and goals that enable or induce the agent's actions or by the explanation of the next action if the current action is in a sequence.

1.3.4 Shortcomings

For satisfying the research goal, the mentioned related implementations were found to be not sufficiently suiting. Although MYCIN was able to explain its reasoning process in English, it could not justify its behaviour. In Chapter 2 it is showed that justifications are minimally required for the user's trust in the system. XPLAIN on the other hand, could justify its behaviour. However, the structure of this expert system is fixed and does not correspond to that of each HeMAS agent. The reasoning rules were hand-programmed while HeMAS mines its own rules. Furthermore, XPLAIN used some sort of console to allow users to communicate with the system. In Chapter 3.2.2, it is shown why this is not believed to be sufficient. Another approach has been the use of explainable BDI-based agents. HeMAS though, does not adapt the BDI-model for its agents and since this approach assumes otherwise, it cannot be applied to the situation of HeMAS in full.

1.4 Outline

This thesis is organised as follows.

Chapter 1 introduces the reader to the topic of this thesis and HeMAS, and presents the goal and research questions. Some former work that show overlap with the goal are mentioned and the applied methodology is presented.

Chapter 2 provides the reader with a brief introduction to a couple of relevant topics. Its first Section treats Explainable AI and all related issues.

The second Section is about Natural Language Generation, another field that is relevant to the research of this thesis.

Chapter 3 applies aspects of its preceding Chapter and provides a suggestion for a solution to the first research question. The first Section describes that the core of the solution is to make HeMAS configurations more transparent by generating explanations for them. The second Section imposes conditions on these explanations. In the third Section, it is proposed to assign a dedicated explanatory component to the actual generating of explanations. The fourth Section proposes the adaption of current HeMAS agents. Section five abstractly illustrates the output that is aimed to.

Chapter 4 discusses the implementation of the solution that has been proposed in its preceding Chapter and thereby addresses the second research question. The first Section gives an overview the solution design in general. The second Section proposes and defends the classification of HeMAS agents. The third Section imposes the need for a model in which each agent presents parts of its reasoning and shows how this model is implemented and should be used. Section four imposes another important issue; the use of an interlingua for the generalisation of the communication of the agents. It proposes the use of First Order Predicate Logic and shows some involved implementation details. The fifth Section describes how sentences of the interlingua, First Order Predicate Logic sentences, can be translated to natural language, English in particular. Section six shows an example implementation.

Chapter 5 describes details of LogicBabelfish, a programming language for the translation of logic to natural language. It is a part of the implementation and therefore also addresses the second research question. Section one gives an introduction to LogicBabelfish. The second Section describes the syntax, including the EBNF. Section three described the semantics with topics like grounding, calling, priorities and other. Section four illustrates how LogicBabelfish can be applied in general and provides some examples for this.

Chapter 6 concludes this thesis. It evaluates the goal and research questions with the work that has been described in the preceding Chapters. The hypothesis is revisited and evaluated as well. Also, some directions for future research are suggested.

The structure of this thesis is sequential; the Chapters follow each other sequentially, with the slightly exception of Chapter 4 and Chapter 5 which can be read interchangeably. In this thesis, for the convenience, all references to persons of an unspecified gender shall be done using male pronouns (to avoid constructions like ‘he/she’, ‘him/her’, ‘his/hers’, etcetera).

Chapter 2

Relevant Literature

This Chapter will provide brief knowledge of relevant issues that require attention for understanding the further Chapters. It is intended to introduce the reader in the material of this research. The first Section covers theory about explaining and understanding of AI systems. The solution presented in Chapter 3 has been inspired and is primarily based upon what is outlined here. The second Section contains material that is more related to the implementation of the solution.

2.1 Explainable AI

Explainable AI is the field dedicated to the automatically generation of explanations for Artificical Intelligence systems. Explainable AI is applied in particular in the fields of Knowledge-Based Systems (KBS), training and tutoring systems or computer simulations. It has been frequently studied in the early field of expert systems, in order to provide explanations along with the advices or diagnoses [Swartout et al., 1991, Wick and Thompson, 1992]
[Swartout and Moore, 1993]. Some examples of Explainable AI systems have been mentioned in the introduction of this thesis: XPLAIN and MYCIN, though MYCIN was not purposely implemented as within the field of Explainable AI. Other applications of Explainable AI involve the explaining of agents e.g. [Johnson, 1994, Harbers, 2011] or AI behaviour in computer simulations
[van Lent et al., 2004].

2.1.1 The Essence of Explanations

Explanations in AI systems are essential for acquiring or maintaining the client's trust [Pieters, 2011, Roth-Berghofer and Cassens, 2005]. The importance of explanations for knowledge based systems is acknowledged by the number of studies in this field, e.g. [Wang and Benbasat, 2007], [Kayande et al., 2009] and [Giboney et al., 2012]. This impresses the imagination, since an advice

or diagnose can only be rationally followed if the information about the formation is sufficient. Advisory or diagnosing systems such as expert systems should provide an adequate level of transparency apart from the final result [Swartout, 1983]. Research even fosters the idea that the existence of an explaining facility alone already increases the user's trust in the output of a system [Anderson and Wright, 1988, Swinney, 1995]. This phenomenon, the increased trust due to mere the availability of explanations, is called the *explanation effect*. All in all, the essence of explanations for advisory systems, such as HeMAS, is hard to overlook.

Explaining the reasoning of an advisory system can serve multiple purposes. [Wallis and Shortliffe, 1982] define four purposes of explaining in the context of advisory systems:

- **Acceptability** Explanations illustrate the logic behind the reasoning of the system and should thereby improve the user's acceptability of the system;
- **Development Support** Explanations support the development of the system itself by providing insight in the program's reasoning and thus make it easier to expose and solve errors;
- **Defensibility** Explanations might persuade the user of the defensibility of the given advice in cases the advice was not expected by the user;
- **Education** Explanations may also serve educational purposes if the user is unfamiliar or hardly familiar with the content beforehand.

Particularly acceptability and development support are sought after for HeMAS. If a system is to benefit from all four fruits of explanations, there are a few requirements to the system. The reasoning process of the system should be adequately represented and the user must be able to somehow browse through its reasoning steps to allow him to examine the logic on different levels of detail. Furthermore, the explanations should be adapted to fit for users with different level of expertise [Wallis and Shortliffe, 1982]. Identifying the client's expertise level is important, since it helps determining the level of detail that an explanation should possess. This should be done with care, as providing too little information as well as too much can diminish the client's trust in the system [Pieters, 2011].

2.1.2 Types of Explanations

Every person has his own way to explain phenomena. Also, every phenomenon requires a certain type of explaining. For example, the explanation of how a fresh cup of coffee is prepared fundamentally differs from the explanation why all windows must be closed in case of fire. The former is more like an

instruction, while the latter is to justify an action. As for expert systems, [Gregor and Benbasat, 1999] derived four different useful types of explanations from [Swartout and Smoliar, 1987] and [Chandrasekaran et al., 1989], listed below:

Line of Reasoning or Trace The type of explanations that describe the reasoning process is called trace or line of reasoning type of explanations. Such explanations answer to the what-question: what did the system do to derive a certain result? These explanations are particularly useful for programmers during the development process.

Example: "Subsequently, I added a small portion of chocolate to the recipe."

Justification or Support Justifications address the 'why-question': why did the system adapt the observed behaviour or choose the performed action? One could regard the justification as an explanation of the line of reasoning, a 'meta-explanation' if you like.

Example: "Adding chocolate to a recipe reduces the salty taste."

Strategy or Control Strategy explanations explain the general behaviour of the system, in contrast with trace explanations which explain one specific case of the system's behaviour.

Example: "I intend to cook with as least artificial flavours as possible."

Terminological Terminological explanations are those that are used to explain the meaning of a term within a certain context.

Example: "Artificial flavours are those that are not extracted from natural ingredients such as herbs, fruits and spices."

It was early announced by [Swartout, 1983] that solely explaining the line of reasoning is found insufficient to most users. [Taylor et al., 2006] acknowledges this by indicating the necessity for also providing the rationale behind a decision or a justification for how some information is known.

2.1.3 Argumentation Theory for Explaining

The Hempel-Oppenheim-schema [Hempel and Oppenheim, 1948] has been long thought of as a sound representation of the nature of human explanations. According to this schema, an explanation would consist of one or more laws and a logical consequence, such that each law is to be explored after which the consequence is deduced logically. However, this 'deductive-nomological' nature does not correspond well with every-day human explanations, neither with most scientific explanations other than in physics [Salmon, 1989]. Rather, a more satisfactory explanation distincts a proof, the logical deduction of a consequence

from a set of premises like the Hempel-Oppenheim-schema, from an argument, a presentation of all information that is sufficient to convince the recipient of the argument in question [Bench-Capon et al., 1991].

Argumentation Theory is the philosophical study that focusses on how humans reach conclusions using logical reasoning. Argumentation theory and especially argumentation for practical reasoning, is relevant for the study of Explainable AI since arguments for actions can be used to explain these actions [Moulin et al., 2002]. Thus, unsurprisingly, many studies in the field of Explainable AI or related refer to Argumentation Theory, e.g.[Bench-Capon et al., 1991], [Gregor and Benbasat, 1999] and [Giboney et al., 2012].

Toulmin Model

An argument is often visualised as a graph, where the head represents the conclusion and the tails represent propositions that support or attack the conclusion. The use of graphs or visualisation of arguments in general may contribute to the understanding of the argument and is called *argument mapping*. An important contribution to argument mapping has been the semi-formal *Toulmin Model* or *Toulmin Structure* [Toulmin, 1958].

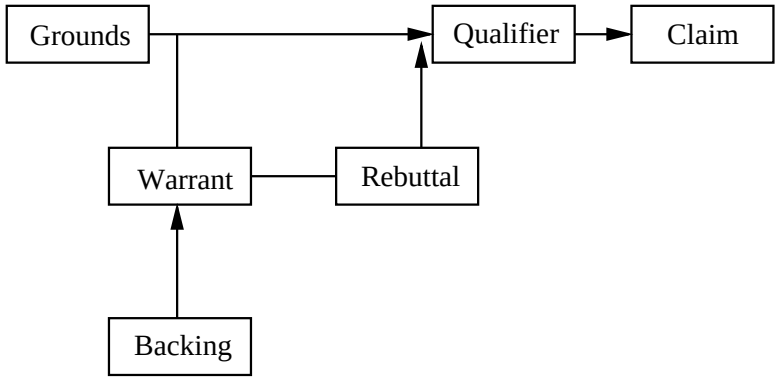

Figure 2.1: The Toulmin Model

Figure 2.1 illustrates the Toulmin Model. It should be noted that the visualisation of the Toulmin Model is slightly divergent among its users, partly because some leave out one or more elements in their visualisation. The Toulmin Model can form a basis for the examination of practical reasoning and argumentation and should make explanations more persuasive, since the model contains elements that are believed to be also present in human argumentation for conviction [Ye, 1990, Everett, 1994, Gregor and Benbasat, 1999]. The Toulmin Model is a semi-formal model, consisting of six dinstinguishable elements:

Claim As usual in Argumentation Theory, arguments are pointed towards a claim or conclusion. The Claim in the Toulmin Model, is that what is to

be defended or argued.

Example: "There is no God."

Grounds The Grounds or Data is[1] the basis of information on which the argument towards the Claim is built.

Example: "There is no evidence for the existence of a supernatural being."

Warrant The Warrant contains rules that bind the Grounds with the Claim.

Example: "Supernatural beings require supernatural evidence."

Qualification Sometimes it is necessary to express a level of certainty of the claim. This is done with the Qualifier within the Toulmin Model.

Example: "Presumable."

Backing The Backing forms an extra foundation on which the Warrant can rest. It contains additional information or an example, making it a supportive element for the Warrant.

Example: "Mathematical claims require mathematical proofs."

Rebuttal The Rebuttal is the element that contains all exceptions to the rule(s) of the Warrant.

Example: "The laws of proof and evidence do not apply to God."

Although there are six elements, not all elements are of equal importance. Obviously, the Claim, Warrant and Grounds are the core of the Toulmin Model, while the Backing, Qualifier and Rebuttal could be left out while maintaining sense for the remaining argument. The coherence of the elements of a complete Toulmin Model can be best understood by concatenating the different elements corresponding to their nature. A possible way to do this would be as follows:

> *"It is QUALIFIER that CLAIM, because GROUNDS and WARRANT given that BACKING, unless REBUTTAL."*

Figure 2.2: The Coherence of the Six Toulmin Elements

Filling in the Toulmin elements in the corresponding labels in the sentence above, obtains a Toulmin Argument that many will find persuasive. The reader is invited to fill in each example provided with each of the Toulmin Model's elements, in the corresponding place in figure 2.2 to see the result for himself.

For a reader with some knowledge in the field of Argumentation Theory, it might be surprising that the Toulmin Model is mentioned, rather than the more

[1] Reference to the Grounds is done in singular form in this thesis, as it stands for a a single element of the Toulmin Model.

recent developments in the field. For instance Prakken and Vreeswijk in e.g. [Prakken and Vreeswijk, 2002] have contributed considerably to the evolving process of Argumentation Theory. Section 4.3 of Chapter 4 describes the reasons for the choice of the Toulmin Model. One reason is its simplicity and hence its understandability.

2.2 Natural Language Generation

Communication is one of the most fundamental properties of the human race. It is substantially communication that allows us to share and practice our ambitions leading to great achievements, like space travel for example. Not without reason Alan Turing has chosen communication as a measure for AI in his Turing Test.

Natural Language Generation (NLG) is a subfield of Natural Language Processing (NLP), Computational Linguistics and Artificial Intelligence. It is the scientific field dedicated to the machine generation of human language. The aim of NLG is to generate grammatically correct, coherent and understandable texts as close as possible to human written texts in a Natural Language (NL) such as English. The counterpart of NLG is Natural Language Understanding (NLU) in which NL is interpreted and transformed to manageable components in a machine language.

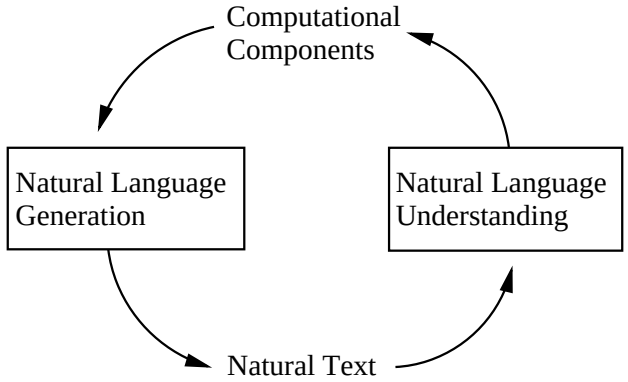

Figure 2.3: Natural Language Processing: Two directions

NLG is a widely used technique in applications such as online assistants for webshops, automatic translation programs, report generators, help message producers, or weather forecast generators. Strictly speaking, even a simple Hello-World program can be regarded as an application of NLG, be that as it may, in its simplest form.

2.2.1 Natural Language Generation Techniques

There are different techniques for NLG. One could subdivide these in three possible forms as below:

Canned Text This is the simplest form of NLG and consists of solely static content. Canned text is written as a whole and thus has no dynamic behaviour whatsoever.

Template Filling A more dynamic method is the automatically filling of text templates. Templates consist of human written text with some variables that allow for the program to ground values and generate dynamic content. This method is used a lot in generating automatic e-mails in companies and universities. Template filling can result in a quite convenient outcome, as was shown by ELIZA [Weizenbaum, 1966], which managed to fool some people and make them believe it was a human.

Grammar-Based Generation The most complex technique is the generation based on the grammar of the target NL. The to be generated sentences are composed from multiple abstract components. Examples of this technique include Functional Unification Grammar, or Functional Discourse Grammar.

One NLG technique is not per se better than another. For each technique there is a situation in which it fits best. Also, complexity might be an issue on choosing a NLG technique, where techniques of which the initial input is closer to the surface text - like canned text - require less effort than techniques with an input more abstracted from the surface text. Meanwhile, the former techniques involve less to no flexibility, where the latter do more.

2.2.2 Architecture of Natural Language Generation Systems

There are several different architectures for NLG systems, but the most frequently espoused is the pipelined architecture, as acknowledged by Reiter and Dale [Reiter and Dale, 2000]. Figure 2.4 illustrates this architecture extended with components that are included in each module of the pipeline [SIGGEN, 2012].

Document Planner The Document Planner receives the communicative goals and is to specify what the output document should be. Here is chosen what is included and what is excluded, a choice that is called Context Selection. For this, the Document Planner takes the communicative goals, a knowledge source, a user model and a discourse history.

Micro Planner The purpose of the Micro Planner is to refine the document plan and produce the text specification more fully. It involves Lexical Selection or Lexicalisation which is the selection of lexical items - a word or a combination of words - for the expressing of meaning, Aggregation

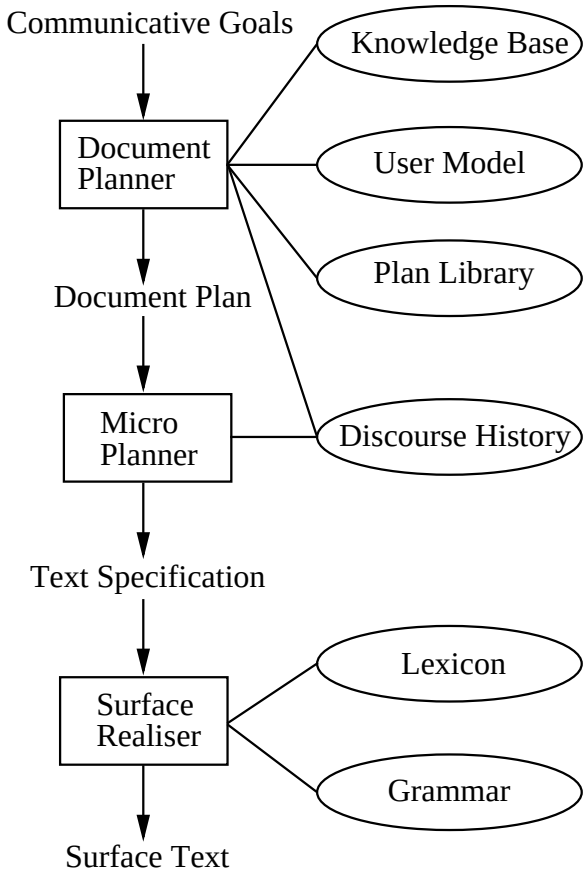

Figure 2.4: Architecture of a NLG system

which is the choice for how sentences are combined to form one coherent whole. Also the determination of the referring to entities is done during the Micro Planning.

Surface Realiser The Surface Realiser does the translation from the abstract notations that were composed by the earlier modules to natural text.

2.3 Chapter Conclusion

This Chapter has provided a background for the sequel of this thesis. It has treated Explainable AI; why explanations are useful for knowledge based systems, in which forms these systems are explained, the application of argumentation theory with the Toulmin Model in particular. It also provided a brief overview of NLG; the three overall techniques and the common architecture for

NLG systems. These matters form the fundaments of the subsequent Chapters, which address the development of a solution for the research goal. The next Chapter defines what the solution should be like in general, while the following two Chapters describe how it is to be implemented.

Chapter 3

Adequately Explaining HeMAS

This Chapter combines the theoretical fundaments of Chapter 2 with HeMAS. Here, the desired situation is presented based on what can be learned from the literature. This Chapter thus answers the first research question. However, implementation details will not be addressed. The next two Chapters will cover the actual implementation of the ideas presented here.

Please recall HeMAS[1], the Heterogenous Multi Agent System functioning as a KBS and the target for this research. It allows one to develop configurations containing agents and other sub-configurations. In this thesis, these agents shall be referred to as respectively *Config Agent* and *Atomic Agent*, for the sake of clarity. For both, explanations are to be generated in order to grant the highly desired[2] transparency.

3.1 Transparency of Configurations

The goal of this research is to make HeMAS more transparent in order to obtain the user's trust. HeMAS can be seen as a tool for building agent configurations that offer advice in a specific medical domain. In fact, it is not HeMAS as a whole that is of relevance to the user in the end, but the HeMAS configurations. The user is probably not interested in explanations about how HeMAS was programmed, yet only in the outcome of a configuration and the reasoning towards this outcome. Therefore, the goal of making HeMAS more transparent, can be reduced to making HeMAS configurations transparent. In the sequel of this document, explaining HeMAS or making HeMAS transparent, should be read as enabling HeMAS to automatically generate explanations for each configuration. The following Section presents the requirements that an explanation for a configuration should meet.

[1] See Section 1.1.1.
[2] See Section 2.1.1.

3.2 Requirements for Explanations

Composing the requirements for an AI system such as HeMAS, consists of the determination of two parts: *what* should an explanation contain and *how* is an explanation presented.

3.2.1 Content

The previous Chapter presented the results of several studies which observed that the MYCIN approach - merely explaining the system's trace or line of reasoning - was not sufficient for obtaining the user's trust in the system. Users also expect justifying explanations that explain the rationale behind the system's reasoning path [Swartout, 1983, Taylor et al., 2006]. Swartout observed that justifications are in fact explanations of the applied rules and are therefore not to be found anywhere in the system, but in the programmer's mind. Unlike the traditional expert systems, HeMAS induces its own rules and should therefore more easily be able to provide justifications for them; a line of reasoning explanation for a rule mining agent should suffice.

Another type deserving investigation is the terminological explanation. Important to note in this, is that there are two kinds of users for HeMAS: physicians and AI researchers. Given their different scientific backgrounds, these groups clearly require different terminological explanations. To be precise, physicians are assumed to require terminological explanations on subjects of AI, while AI researchers are assumed to want those explanations on medical subjects. Section 3.2.2 implies why the realisation of making these divergent terminological explanations available, is not that hard.

Configuration Overview

When explaining a sequential plan that includes symbols and connections with lines or arrows, it is not unusual to first explain the plan in general; describing the plan while explaining what each part means. At first, this seems also applicable to HeMAS configurations. Such a general explanation contains much trivial information though. Take for example the following general explanation for an imaginary configuration: "this configuration contains two agents, such that agent a is connected to agent b, agent a initialises the configuration and agent b concludes it". This information is quite trivial and can easily be observed by taking a glance at the visual configuration. Another, even more general explanation could be: "a configuration consists of several agents that are connected to each other, meaning that the output of one agent forms the input for an other agent". This explanation is actually useful, since it intends to clarify how HeMAS configurations work.

However, providing this or a similar, static, general explanation with each configuration would result in repeating the same information with each explanation of a different configuration run, which might be annoying to users. It makes more sense to include such general information in a manual provided

with HeMAS. The HeMAS explanation component should not include general non-configuration specific information.

3.2.2 Presentation

It is customary for Explainable AI to generate explanations in an NL[3] for the simple reason that humans explain in NL also. NL is more commonplace than diagrams and is more self-evident, while the problem with sole diagrams is that their interpretation may be ambiguous. Furthermore, textual messages can be more accurate in their expression than diagrams. In the case of HeMAS, we are in a somewhat luxury position, as some of the explainable components, the config agents, already have a visual diagram that may partially explain its behaviour. The choice for explanations in NL seems therefore logical. In this thesis, the English language is primarily focussed on, considering the timespan for this research and the infeasibility of equally addressing several languages within this timespan.

Another important aspect of explaining is *interaction*. Argumentation Theory treats explaining as a dialogue and this seems to correspond with human explanations. It often appears that during (or after) the explanation of an event by a person, the listening person asks a question on certain parts of the explanation of which he feels it was lacking information. It should therefore be made possible for the user to somehow steer the explanation of HeMAS in a way that fits the user. By doing so, the user can decide how much information he wants and the danger of providing too much or too less information, as stated in [Pieters, 2011], is avoided. Allowing the user to browse through the explanation is one of the requirements indicated by [Wallis and Shortliffe, 1982]. The early expert system XPLAIN acknowledged this by allowing the user to type a few predefined words. Though sufficient and maybe advanced for that time, this solution seemed not very elegant for a modern KBS like HeMAS. Firstly, the user must somehow get to know these reserved words. But, as shall be clarified later, not all information is available at all time, so the list of possible words varies per explanation. Secondly, how should the system respond to the incorrect use of words? Capturing and correcting small errors in the typing is nice of course, but requires study *an sich*. Thirdly, selecting the type box followed by using the keyboard to type and return the keyword can be too much of an effort for a user, while studies show that the likelyhood that an explanation is actually used, is closely linked to the ease to access the explanation [Gregor and Benbasat, 1999, Mao and Benbasat, 2001].

A 'click-through hyperlinking mechanism', as is very common on the web, seems a more appropriate solution. It requires small technical effort, yet allows the user to request more information on a specific subject and since many people are used to this mechanism, thanks to the popularity of websites like Wikipedia, it should make the explanation facility of HeMAS accessible.

[3] See Chapter 2.

3.3 Agent Clara

Now that the requirements for the to be generated explanations are formed, another important issue is to determine how the explanations generating component is to be fitted into HeMAS. Clearly, to meet the mentioned requirements, all existing HeMAS agents should be adapted to *explainable agents*. A messy approach would be to make each agent fully explain its own behaviour. This solution would require a dedicated explanatory NLG component for each agent. This is highly redundant, not condictive to the maintainance of the system and therefore undesirable. Rather, a single component collecting or accepting all necessary information from the agents that is responsible for the generation of the explanations seems a much better solution. Furthermore, assigning an *agent* to the role of the explanatory component would nicely fit the notion of HeMAS as multiagent system. More importantly, there are some advantages for the creation of an *explain agent* apart from the design technical beauty:

- One advantage that already was mentioned more or less is the better maintainability for the explaining component. Making this component an agent will also make it distinct, allowing developers to repair or improve it more easily.
- Also, the adjustability for the generated explanations for a configuration will be improved. Including and excluding agents from the explanation can be simpler and more natural.
- Another advantage is that HeMAS can be easily extended with multiple agents that descent from the original explain agent, such that different (new) requirements can be met. For example, different explain agents can be developed to support different languages or different clients.

In HeMAS, it is customary to baptize an agent with a name. Since the explain agent's purpose is to clarify behaviour, a nice name would be *Clara*[4].

3.4 Explainable Agents

Apart from developing a component for generating explanations, the existing HeMAS agents need to be adapted such that they provide all information necessary for Clara to sufficiently explain. There should be an agreement or protocol between Clara and the (other) HeMAS agents that involves the information that Clara is to acquire. It is, so to say, necessary for HeMAS agents to become *explainable HeMAS agents* which means that these agents guarantee that information about their reasoning process is supplied to Clara in an established form.

[4]Derived from 'claro', which is Latin for 'to make clear'.

3.5 Example Output

Now a fictional output will be presented. Figure 3.1 is to represent a simple HeMAS configuration.

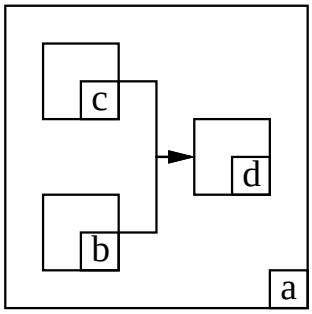

Figure 3.1: Example HeMAS Configuration

Assume that this configuration is to be explained, or rather that config agent a is to be explained. The boxes b, c and d are also agents. According to the previous Sections, the output will look somewhat like the following:

"d concludes x, because b concludes y and c concludes z and b and c are the basis of d"

Figure 3.2: Example Fictional Abstract Explanation

The underlined parts represent clickable areas that will cause the generation of another corresponding explanation. Clicking on d, b or c separately, will cause a terminological explanation to be generated, giving some global information about the agent. For example, if agent c is a neural network, some information about neural networks along with references is provided. Clicking on *concludes x* or *concludes y* will result in the trace explanation of the reasoning of respectively d and b. Such an explanation is similar to the one in figure 3.2. A justifying explanation is generated by clicking on *b and c are the basis of d*, since it explains why this construct is what it is.

3.6 Chapter Conclusion

In this Chapter, conditions on the component that will be responsible for explaining HeMAS were set. The explaining component is called Clara and will

be an agent for improved maintainability, adjustability and extensibility. Meanwhile, all existing HeMAS agents must be adapted to explainable agents and provide information conform an agreement or protocol. Agent Clara must generate explanations that provide tracing explanations as well as justifications and explain terminology where this is found necesessary. The explanations must be presented as such, that the consulting user is able to request for more details by means of a click-through mechanism. Meeting these conditions is believed to improve the transparency of HeMAS and acquire the trust of the client. The example output illustrates what is aimed for. With this, the first research question has been addressed.

The next two Chapters address the actual implementation design of the solution.

Chapter 4

Design

This Chapter addresses the actual solution design of what has been presented in the previous Chapter. It therefore applies to the second research question concerning the technical realisation of automatically generating explanations. In this chapter, Java is assumed as the primary programming language for the implementation of the proposed solution.

The first Section gives an overview of the solution that this Chapter presents. The second Section discusses the necessary classification of agents. The third Section introduces the use of the Toulmin Model to explain steps in the reasoning process of an agent. The fourth Section presents the form in which explainable agents should communicate to the explaining component Clara, while the fifth Section substantiates this. Section six shows an example implementation.

4.1 Overview

Figure 4.1 gives an overview of the solution that is presented in this chapter. Here the reader is introduced to the solution briefly, but all details are presented in later Sections in this Chapter.

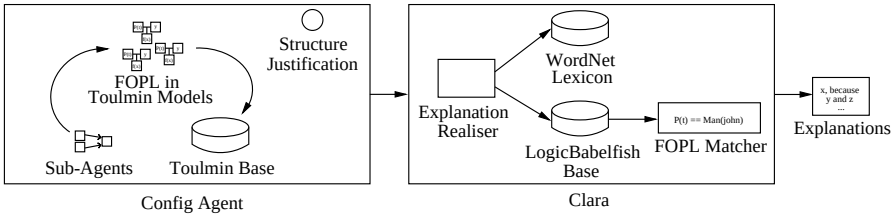

Figure 4.1: Solution Overview

Clara is connected to a Config Agent, which is the HeMAS configuration that is to be explained. This Config Agent contains one or more Sub-Agents (which

can be Atomic Agents as well as Config Agents if the reader recalls the recursive nature of HeMAS). These Sub-Agents should provide information about their reasoning by means of First Order Predicate Logic (FOPL) in Toulmin Models per transition. This modelled information is stored in a Toulmin Base, which grants Clara to reconstruct the reasoning process in NL. On top of explaining the individual Sub-Agents, their relationship, which forms the structure of the config Agent, also requires explaining. The provision of this structure justification is howver not within the scope of this research.

Clara contains an 'Explanation Realiser', which is a main controlling system that calls a LogicBabelfish Base and generates the final explanation. Also, Clara determines the equivalence of logical forms, since LogicBabelfish cannot do this internally. The reader is invited to examine the details of these concepts in the Sections that follow.

4.2 Classifying Agents

Although in this thesis the strive is towards a solution that is as general applicable as possible - meaning that different agents are treated as equally as possible - there still is the necessity to classify agents into different sorts. Recall that HeMAS contains quite differently natured agents of which the functionings can hardly be compared. Take the Bayesian Network Agent (Probability Theory) and the Boltzmann Machine Agent (Neural Network) for example. The outcome of the former can be explained by tracing each reasoning step, while the latter uses technology that almost prides itself on its mysterious, unexplainable reasoning. It therefore seems relevant to distinguish two classes of agents: Limpid Agents and Cloudy Agents.

Limpid Agent A Limpid agent is an agent of which the reasoning can be divided into a sequence of clear steps. Each clear step could then be part of the generated explanation (although not every step is initially reviewed, in accordance with the click-through-mechanism). The step size should be such that each step should contribute to the comprehension of the Limpid Agent's behaviour. An (already mentioned) example AI technique that an agent can apply for the classification of Limpid Agent can be the Bayesian Network. Other examples include logical deduction or tree search algorithms.

Cloudy Agent Any agent that cannot be classified Limpid agent on the other hand, is a Cloudy agent. Most Cloudy agents are subsymbolic agents, such as neural networks. But also agents that apply statistical techniques such as regression analysis, can be classified as Cloudy agents.

One can determine whether an agent is Limpid or Cloudy by asking oneself the question: can the reasoning of the agent be divided into distinct steps such that if each step is explained to a person not familiar to the applied technique, the person's comprehension of the agent's behaviour is improved? If so, the agent could be classified Limpid agent; Cloudy agent otherwise.

4.2.1 Approaches

In Section 3.2.1, it was stated that a line of reasoning amplified with justifying explanations are desirable for sufficiently explaining. According to the previously defined agent classifications, these explanations can be generated for Limpid agents, but unfortunately, not for Cloudy agents. The reasoning proces of a Cloudy agent is by definition vague and is not meant to be understood, other than very broadly. Cloudy agents therefore require a slightly different approach than Limpid agents if they are to be explained.

It should be clear that the user of HeMAS, as a physician, is not interested in technical details that concern the outcome of the reasoning of Cloudy agents. Yet, also Cloudy agents require sufficient trust from the user and should therefore be explained in at least some way. In Section 3.2.2 of the previous Chapter, it was stated that explanatory information should be available on demand. It therefore seems appropriate to refer to information sources for the techniques of Cloudy agents. Please recall the explanation effect[1]; providing an explanation alone should increase the user's trust. So even if the user will not examine the references, his trust in the system should be increased by the observation that there exists sufficient (scientific) information that explains the AI technique of a Cloudy agent. Briefly, a Cloudy agent must provide the following information:

- The outcome of its reasoning;
- References to (preferable scientific) articles that explain or defend the use of the Cloudy agent's techniques;
- Actual tests that proof the success or failure[2] of the AI technique;
- Any other information that might defend the agent's reasoning.

In order to reconcile the treatment of Limpid and Cloudy agents as much as possible, a Cloudy agent is explained as what could be called a 'black-boxed' Limpid agent; a Limpid agent with non-transparent parts. More specifically, Cloudy agents should be treated as Limpid agents with one single *transition* providing the above described information. Section 4.3 focusses on transitions further.

Config Agent

The Config Agent can be classified as a Limpid Agent, because the connections between the agents can represent the transitions, and explaining these is believed to improve the user's comprehension of the Config Agent. Unfortunately,

[1] See Section 2.1.1.

[2] It might seem funny to the reader to also mention unsuccesful tests here, but as has been mentioned, the purpose of the provision of this information is to gain the user's trust in HeMAS and certainly not to convince the user to abide by the given advice. It therefore is necessary to be 'honest'; to also provide the significant weaknesses of the used AI technique if there are any.

the justifications for the structure of a Config Agent cannot be automatically generated, because this structure is originated in the minds of the medical and AI expert that invented it. For this reason, if the Config Agent is to be explained in full, these experts are required to provide justifying information with each transition they compose.

As the goal of this thesis is to find a solution to automatically explain HeMAS, inventing a protocol that Config Agent composers should adhere to is outside the scope of this thesis and is thus a possible direction for future work. When finding a solution for the justification of Config Agents, it should be taken into consideration that using canned text or simple text templates might undo the generic character - the possibility to generate different languages for example - of the current solution for generating explanations.

4.3 A Model for Transitions

In order for Clara to be able to explain the line of reasoning of a HeMAS agent, she[3] must somehow be programmed to know how each step of the reasoning process of an agent should be translated. This task is not straightforward, given the heterogeneous property of HeMAS. In HeMAS every agent applies a different AI technique and thus requires a different methodology to translate it to an explanation in NL. It is infeasible to program Clara such that she translates these different techniques all independently. Furthermore, this would cause a weak maintainability for Clara, as possible future changes to the requirements for explanations would require Clara to adapt each explanation generation mechanism for each explainable agent.

One step in the reasoning process of an agent is comparable to a transition between two states. Roughly, an agent performs a step from reasoning state s to s' if a certain condition c is met:

$$s \xrightarrow{c} s'$$

Here, c can be anything that the explainable agent applied to go from reasoning state s to s', also s and s' can be any representation of knowledge. Clara must at least obtain s, c and s' if she is to explain a reasoning step. Yet, obtaining $\{s, s', c\}$ still requires Clara to apply agent-specific knowledge, for she must recognise and distinguish the states from the condition. To avoid this, it is required to use one model in which the explainable agents 'store' their information concerning a transition. The Toulmin Model [Toulmin, 1958] has been suggested to be applicable in Explainable AI [Bench-Capon et al., 1991, Ye, 1990, Everett, 1994] [Gregor and Benbasat, 1999], since it supports all different aspects that are required for explaining a transition. On top of that, it corresponds with the manner in which people use arguments and explain phenomena[4]. Practically, Toulmin Models can be easily implemented and their

[3] Although Clara is an agent and not a person, I prefer to refer to Clara with 'she'.
[4] See Chapter 2.

content can be easily translated to NL as we shall see later in this Chapter. Another nice aspect of Toulmin's Model is its simplicity; its structure is not hard to understand, saving explainable agent developers the time of learning and applying complexer models. Also, due to its simplicity, the generated Toulmin Models might also serve as illustrations provided along with the textual explanations in future extensions of Clara.

4.3.1 Applying the Toulmin Model

As was described in Chapter 2, a Toulmin Model contains at most six elements: claim, qualifier, grounds, warrant, rebuttal and backing. The backing exists only to support the warrant, so technically this element could be replaced by another Toulmin Model such that the warrant of the original forms the claim of this new model. Since the idea of linking Toulmin Models (as shall be explained in the next paragraph) will be applied for Clara, the use of the backing is no longer of use and will be excluded from the Toulmin Model applied in this implementation. Hence, only five elements of the Toulmin Model remain and therefore in the remaining of this thesis, it shall be referred to as a *Restricted Toulmin Model*.

Explainable agents must create a Restricted Toulmin Model for each of their transitions. Each element will hold a part of the information about a transition, such as the former state, the applied rules and the latter state. Though this model can consist of up to five elements, not all elements are obligatory; a Restricted Toulmin Model is sufficient if it at least contains a claim, grounds and warrant. The qualifier and rebuttal are both optional. The following summation defines how each Restricted Toulmin element should be instantiated per transition for a Limpid Agent:

- The claim must hold the succeeding state or the conclusion;
- The grounds must hold the preceding state;
- The warrant must hold the conditions that were applied to the preceding state;
- The qualifier may hold a degree of certainty;
- The rebuttal may hold possible exceptions.

For a Cloudy Agent, it is less important to use strict rules concerning the use of the Toulmin Model than for a Limpid Agent. There is no line of reasoning that should somehow be extracted from the Toulmin Model provided by a Cloudy Agent. Rather, one explanation that supports the AI technique is what is aimed for to explain an agent of this particular class. Nevertheless, a directive for how Cloudy Agents should apply the Toulmin Model would not hurt, so the following application of the Toulmin Model is proposed for a Cloudy Agent:

- The claim must hold the conclusion of the agent;

- The grounds must hold the (most important) premises;
- The warrant must hold the supportive references and/or tests;
- The qualifier may hold a degree of certainty;
- The rebuttal may hold side notes or references that question the applied technique.

It is believed that a programmer of an explainable agent should be able to decide how a transition of the agent fits best in the Restricted Toulmin Model.

Example 1:

This example shows how a transition of an explainable agent that uses a Bayesian Network can be fit into a Restricted Toulmin Model. It is important to keep in mind that this 'model fitting' is a matter of choice and could possibly have done differently.

In Bayesian Networks, $P(x)$ denotes the probability that x holds and $P(x|y)$ denotes the conditional probability of x given y. In this example, the following prior probabilities are known:

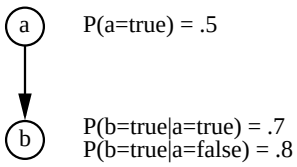

Figure 4.2: Example Bayesian Network

Now let the deduction of $b = true$ be the transition for this example. The probability of $b = true$ can be calculated as follows:

$$P(b = true) = .7 * .5 + .8 * .5 = .75$$

The Restricted Toulmin Model in figure 4.3 would suffice.

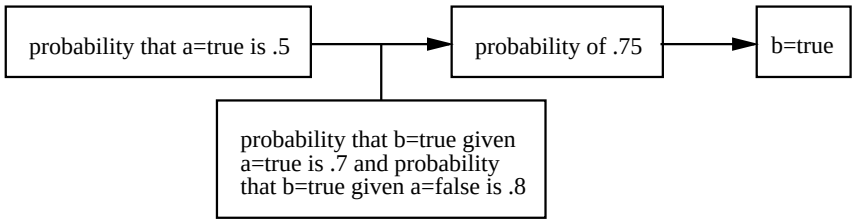

Figure 4.3: Example Restricted Toulmin Model for Bayesian Transition

Notice that the Restricted Toulmin Model contains elements in natural English, this has been done for the illustrative purpose only. Section 4.4 covers what actual form these elements should have.

4.3.2 Toulmin Chain

One of the requirements that were formed in Chapter 3 is the click-through mechanism; the user should be able to click on elements of explanations and by doing so, obtaining further explanation on demand. This functionality can be achieved by linking Toulmin Models to what in this thesis is referred to as a *Toulmin Chain*. Toulmin Chains are concatenated Restricted Toulmin Models, such that the claim for model m equals the grounds, warrant, qualifier or rebuttal for model m'. Figure 4.4 illustrates this principle.

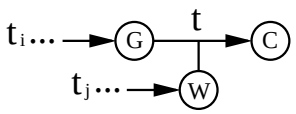

Figure 4.4: Toulmin Chain

In this abstract minimal Restricted Toulmin Model t, C has the role of claim, G the role of grounds and W the role of warrant. While G is the grounds of model t, it is also has the role of claim in another Toulmin Model, t_i. Similarly, W is the warrant of t, yet also is the claim of t_j. This is the idea of a Toulmin Chain. Practically, if a user requests more information about the warrant (a justification question), model t_j is presented as a response.

From Figure 4.4 can be observed that claims are uniquely bound to a Toulmin Model, as C is bound to t in the figure. A claim can be considered as the key element of a Toulmin Model since all elements point towards it. In other words, a claim is supported by a Toulmin Model. This is an important observation for the implementation of Toulmin Chains, as we shall see next.

4.3.3 Implementing Toulmin Chains

One Toulmin Model can simply be implemented as a class with accessible attributes that correspond to the elements of the model. So for a Restricted Toulmin Model, that would be five attributes to cover the Claim, Grounds, Warrant, Qualifier and Rebuttal. Toulmin Chains are implemented differently. A Toulmin Chain can consist of Toulmin Models generated by different agents. For this reason, it is not practical to use a datatype for the Toulmin Chain explicitly, as different agents would have had to contribute to the same instance. A much simpler approach is to use some sort of database that implicitly defines Toulmin Chains by means of their elements. The Toulmin Models are then chained in the same way rows are linked with keys and foreign keys in SQL. These Toulmin Chains can then be retrieved model by model.

This approach should work fine, because the Toulmin Chains are only of use during runtime when the user clicks on an element and requests more information. Figure 4.5 is a class diagram that shows how Toulmin Chains can be implemented (implicitly).

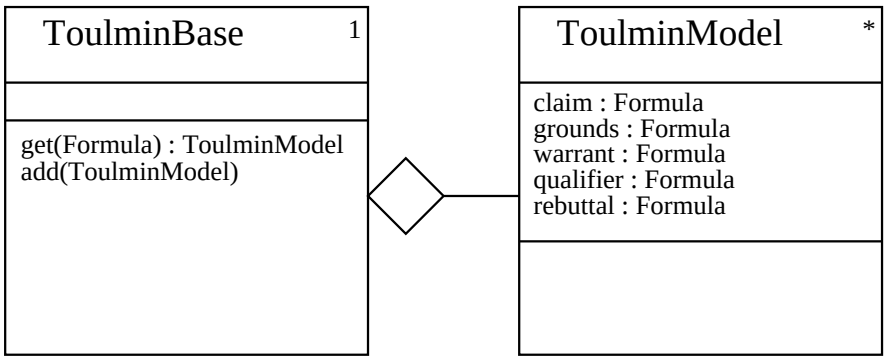

Figure 4.5: Class Diagram for Toulmin Model

The ToulminBase is a singleton class that functions as a database for storing ToulminModels. This storage can be done by means of a HashMap, of which the value is a ToulminModel and the key is the Claim of that ToulminModel. Every newly created ToulminModel object should immediately be registered to the ToulminBase.

4.4 Interlingua for Explanations

For the elements of the Toulmin Models that are to be generated by each agent for each transition, it is necessary to specify in what form the information is to be stored. The goal of the Toulmin Models is for Clara to be able to generate explanations from the information in these models. It follows that the information must be in a form that is understandable for Clara. Clearly, one standardised form is required that all agents can and will adhere to when generating Toulmin Models. Otherwise, each agent would still require its own NLG component, which is not conform the requirements of the wanted solution[5]. From here on, this singular form shall be spoken of as an *interlingua for explanations*, abbreviated to interlingua. There are some requirements for the choice of an interlingua within this context:

Expressiveness The expressiveness property determines how well an interlingua can be used to express phenomena in NL.

[5]See Chapter 3.3.

Propinquity With propinquity is meant how close the interlingua of choice is connected to any explainable agent. A close interlingua would be a form that requires little effort to get to from the original agent reasoning form.

Acceptability The acceptability determines how well the interlingua is accepted by the agent programmers. An agent programmer must be able to adapt his agent to fit the interlingua and therefore the interlingua must be acceptable for him.

Clearly, the choice of interlingua depends on compromising between these requirements, as practically no interlingua can fit each completely and to an equal degree.

In Chapter 2.2.1, Grammar-Based Generation was mentioned as an NLG technique. One advantage of the use of Grammars (like Lexical Functional Grammar, Head-driven Phrase Structure Grammar, Functional Discourse Grammar or other) is its wide expressiveness. Grammars have been designed considering divergent possible sentence constructions. They allow a great flexibility and are therefore perhaps the candidates offering the most expressiveness available for NLG today. Unfortunately, for this expressiveness one pays the price of the greater complexity these Grammars involve. It is due to this complexity that Grammars are not found convenient as an interlingua; it would not suit well with the acceptability requirement.

Prolog is a candidate that also deserves some consideration as it has been used in other NLG studies (i.a. [Bench-Capon et al., 1991]). Prolog is formal and would therefore not be too difficult for explainable agents to adhere to. The acceptability is also no reason to drop this candidate, as Prolog is well-known under AI scientists and fairly easy to learn. The expressiveness though, leaves something to be desired. For instance, Prolog embraces the *closed world assumption* - the assumption that the unknown does not hold - with the result, the absence of negated facts. Sentences like 'the patient has no headache' are thus not possible (or at least not using the standard constructions that Prolog offers in a clean way). A more logical[6] choice is the superset of Prolog: First Order Logic.

4.4.1 First Order Predicate Logic

One salient candidate for an interlingua is FOPL, because it is not uncommon to use this formalism for NLG, e.g. [Wedekind, 1988], [Shieber et al., 1990] and [Mpagouli and Hatzilygeroudis, 2007]. If FOPL is to be compared to the three requirements of the previous Section, it can be found that its expressiveness is fairly sufficient for the purpose in mind due to the declarative character of FOPL. The formal character of FOPL contributes to its propinquity; it should not be too difficult to translate a machine reasoning process to a formal language as the machine language itself is formal. The acceptability of FOPL is actually

[6] Pardon the expression.

quite attractive, for it is widely known within AI and thus should be so for agent programmers as well.

4.4.2 Implementing Logic

If FOPL predicate logic is used as interlingua, it is necessary to build a parser that decomposes formulae that are given as input and generates computable objects. Observe that a well-formed formula (WFF) is either atomic - with no sub formulae - or a concatenation of two WFF's with one connective (e.g. and, or, implication), or a quantifier concatenated before a WFF. All formulae that are well-formed and have more than one connective, can in fact be rewritten with parentheses such that the stated observation holds, for example $x \wedge y \wedge z$ equals $(x \wedge y) \wedge z$. This allows one to create a datastructure in tree form as in figure 4.6.

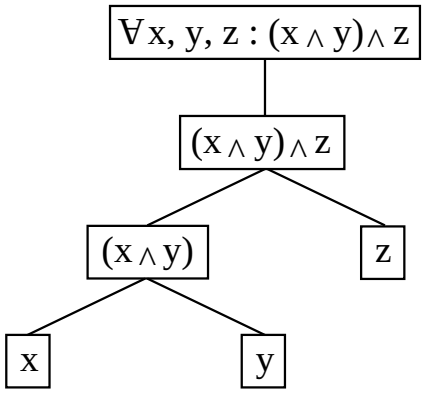

Figure 4.6: Example Logical Formula Decomposed

The concept that is visualised in figure 4.6 can be implemented nicely in a Object-oriented programming language.

4.5 Translating First Order Predicate Logic to Natural Language

An important aspect of Clara is the translation of the information that was obtained from the explainable agents to a western NL. As has been stated earlier, this information is offered in Toulmin Models, where each element of this model contains the information in FOPL form. In other words, Clara is required to translate FOPL to NL explanations. Figure 4.7 shows the architecture of Clara that allows her to translate FOPL to NL.

In this Section it is shown how FOPL can be translated to NL and what tools are required for this. Here it is assumed that FOPL formulae constist of operators (e.g. logical and, or, implication, ...), functions and predicates [7].

[7] Quantifiers are discussed in the next Chapter.

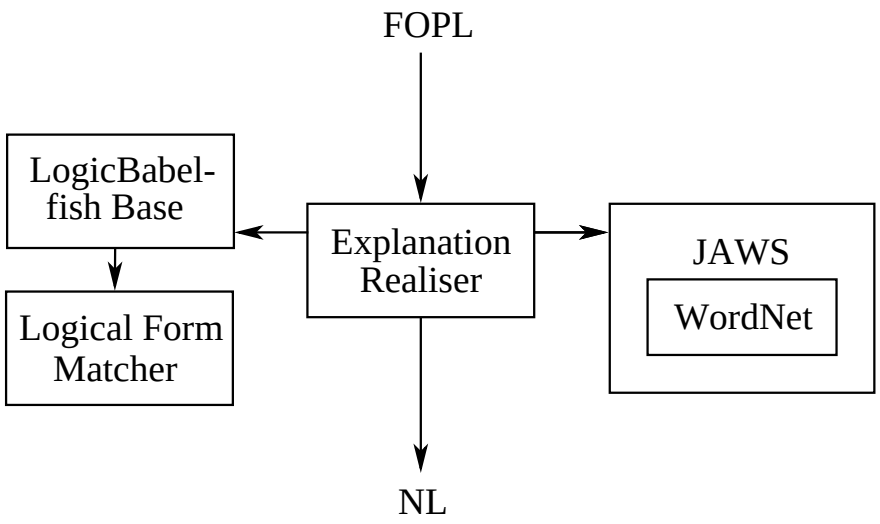

Figure 4.7: Translating FOPL to NL

4.5.1 Operators

Logical operators are the connectives between WFF's. Their translation is quite straightforward. Table 4.1 shows a few translations of very common operators. There are of course many more operators, but their translations are trivial and therefore need not be listed.

operator	**Interpretation**
$x \rightarrow y$	if x then y
$x \wedge y$	x and y
$x \vee y$	x or y

Table 4.1: Interpretation of Word Types for Predicate $P(t)$

4.5.2 Functions

A function is a construction that 'returns' a value corresponding to the argument(s) that is/are provided. For example $sqrt(9)$ would be a function that returns the squareroot of 9. Interpreting functions is rather trivial. Let $f(x)$ be a function, then its Interpretation would be 'the f of x'. So $sqrt(9)$ would be 'the squareroot of 9' and $father(jim)$ would be 'the father of jim'. When a function contains more arguments, they are simply concatenated in the Interpretation, so $f(x_1, ..., x_n)$ would be 'the f of x_1 and ... x_n'.

Functions are useful to describe an exact phenomenon. For example:

$$sum(2, 4) = 6$$

This translates to 'the sum of 2 and 4 is 6'. As was stated before, functions can also be nested, allowing constructions such as the following:

$$sum(sum(1, 2), multiplication(3, 4)) = 36$$

The above formula would be translated somewhat like 'the sum of the sum of 1 and 2 and the multiplication of 3 and 4 is 36'. Translating predicates requires more effort, as the next sub Section will show.

4.5.3 Predicates

The translation of FOPL consisting of predicates to NL seems not very difficult at first sight. It is perhaps even natural to try and translate FOPL formulae to NL when reading a text that contains such formulae. A noticeable observation is that different people are inclined to the same translation of FOPL to NL and NL to FOPL. There seems to be some kind of consensus to which one adheres when translating between FOPL and NL. For example, most people agree that the correct translation of $Man(john)$ would be something like 'john is a man', while $Happy(john)$ would translate to 'john is happy'.

Observant readers probably already have noticed the core aspect of translating a predicate with a form such as $P(t)$: it is the determination of the word type of P. The formulae of the previous paragraph for example, have a noun and an adjective word type, resulting in the translation of 'is a' and 'is' respectively. There are three main word types: noun, verb and adjective. Table 4.2 shows the word types and their different interpretations for formulae with one argument.

Word Type	**Interpretation**	**Example**
Noun	t is a P	john is a man
Verb	t P	mary laughs
Adjective	t is P	penelope is happy

Table 4.2: Interpretation of Word Types for Predicate $P(t)$

Readers that are familiar with logic probably recognise the translations from the table, because most people translate between FOPL and NL as shown here. Other translations on which most people agree, are the predicates with two arguments. Table 4.3 illustrates how predicates with two arguments are commonly translated.

So predicates with one argument or two arguments are interpreted often as the tables show. There is no such consensus for predicates with more than two arguments. They simply do not occur as much and the relations between the arguments are often ambiguously interpretable.

Word Type	Interpretation	Example
Noun	t_1 is a P of t_2	john is a friend of rob
Verb	t_1 P t_2	mary tickles fluffy
Adjective	t_1 is P than t_2	penelope is taller than zoey

Table 4.3: Interpretation of Word Types for Predicate $P(t_1, t_2)$

From what has been presented in this Section follows that there is need for the application of a lexicon. A lexicon offers information about words, such as the word types and other. The importance of a lexicon is dependent on the selected target NL. For example, if the target NL would be German, the gender of words would be an important aspect; $Stadt(heidelberg)$ translates to 'heidelberg ist eine stadt', while $Auto(mercedes)$ would be 'mercedes ist ein auto' (notice the 'e' after the article in the first translation). Since English is the primary language for this research, an English lexicon like Wordnet would suffice.

4.5.4 Wordnet as Lexicon

Wordnet is an open virtual dictionary for, in the first place, the English language. There is though much independent work in progress to develop Wordnets for all kind of languages including Dutch, Japanese, Turkish, Russian, Arabic, German and many more. The original (English) WordNet contains more than 125000 words categorised in noun forms, verb forms, adjective forms and adverb forms, and the database is still growing [Fellbaum, 1998, WordNet, 2012]. It allows one to inspect words; determining the word type[8], obtain synonyms, example sentences and more. WordNet has been an active project since its foundation in 1986 [Miller, 1986]. The size of the database of WordNet and its currentness, is believed to make it an excellent lexicon.

Another nice additional of WordNet, is the existence of an API for Java. This API is the Java API for WordNet Searching (JAWS) by Brett Spell [Jaws, 2012], an open source Java project with a size of fifty classes. JAWS serves as a shell for WordNet, which means that WordNet needs to be installed and JAWS requires a reference and access to WordNet. JAWS has been made compatible with the newest version(s) of WordNet which, at the moment of this writing, are WordNet 2.1 and WordNet 3.0.

Following from the previous Section, one aspect that is looked for in a lexicon is the determination of word types. With JAWS this can be done as follows:

[8]For words that can be of different word types, such as love (which can be both in noun form and verb form), one can request the word type that is most frequently used.

Algorithm 1 Determining the Word Type

```
1: public SynsetType getWordType(String word)
2: {
3:    SynsetType result = SynsetType.NOUN;
4:    int probableWordTypeID = 0;
5:    for(Synset synset : database.getSynsets(word))
6:      for(String wordForm : synset.getWordForms())
7:        if(synset.getTagCount(wordForm) >probableWordTypeID)
8:        {
9:          probableWordTypeID = synset.getTagCount(wordForm);
10:         synsetType = synset.getType();
11:       }
12:   return result;
13: }
```

The above code returns the most probable word type of a string. The most probable because many words are ambiguous, for example 'mortal' can be both a noun and an adjective. What is missing in the code above, is the localisation of WordNet and the instantiation of the WordNet database. This can be done as follows:

Algorithm 2 Localisation and Instantiating WordNet

```
1: System.setProperty("wordnet.database.dir", pathToWordNetDict);
2: database = WordNetDatabase.getFileInstance();
```

4.5.5 General Application

The previous Sections have shown how FOPL - operators, functions and predicates - can be translated to NL and how WordNet by means of JAWS can be applied for the determination of word types. As of yet, nothing to actually apply these mechanisms and perform the NLG has been described though.

One approach would be to use text templates for i.a. the translations shown in the tables 4.1, 4.2 and 4.3. Jaws would be used to determine word types and the predicates are translated correspondingly by selecting the appropriate template, while the operators are separately translated using another template.

This whole approach can be directly implemented in Java. One could apply HashMaps to allocate a NL translation to a logical operator for example. The translation of predicates can be done by a dedicated Java method that uses JAWS to lookup the word type of the identifier of a predicate and returns a corresponding NL translation. This implementation would be feasible, yet not very maintainable and expandable as every change or addition would require modifying the Java code. Particularly the maintainability and the expandability demotivate the embedding of any canned text in Java code. A more efficient

solution is to define the templates externally, which is what Figure 4.7 illustrates and what the next Section covers.

4.5.6 External Templates and LogicBabelfish

A possible format for external templates and, due to its popularity a logical consideration perhaps, is *XML*. It could be used to define translations for logical formulae in a structured fashion. The translation would then simply include finding a matching translation for a formula and when found, search and replace corresponding to the definition of the template. Figure 4.8 illustrates how XML could be used for defining translations.

```
<translation language='english' wordType='noun'>
   <formula>
     P(t)
   </formula>
   <nl>
     t is a P
   </nl>
</translation>
```

Figure 4.8: Example Usage of XML for Templates

One disadvantage of the use of XML is that complete templates can become quite large and difficult to supervise. Another more important issue that applies also to any other non-dynamic template format, is that there are logical constructions that cannot be translated to NL so trivially by simply replacing each part separately. For example $Man(x) \rightarrow Mortal(x)$ translates to "if x is a man then x is mortal", while a formula like $\forall x : Man(x) \rightarrow Mortal(x)$ translates to "every man is mortal", which is fundamentally different yet contains the same logical operator ($\rightarrow$). To this end, a more dynamic solution for defining translations is desirable.

LogicBabelfish has been invented with exactly this goal in mind: to allow one to dynamically define translations for logical formulae in a single format for a better maintainability and extensibility. It is also believed to support multiple (western) languages by its flexible nature. LogicBabelfish allows one to program translations for, in the first place, logical formulae. The example formula of Figure 4.8 can be written in LogicBabelfish as follows:

define $\_[P(t)](English, Noun)$
  $t +$ " `is a` " $+ P$ `.`

This construction seems more convenient due to its simplicity. Also, other constructions involving quantifiers are implementable in LogicBabelfish, as shall be clarified in the next Chapter.

One could question the use of templates for NLG stating that template-based NLG is too simple or that it lacks proper theoretical foundation and linguistic insights. [van Deemter et al., 2003] criticise this view and state for example that the differences between template-based NLG systems and other NLG systems have become increasingly blurred. It is though discussable whether a *LogicBabelfish-based* NLG application can be strictly called an instance of template-based NLG. Taking the definition of Reiter and Dale in consideration, template-based NLG systems map their non-linguistic input without intermediate representations to surface text [Reiter and Dale, 2000], while this is not necessarily the case with a LogicBabelfish-based application.

4.6 Example Implementation

As a proof of concept, a small example reasoner has been implemented and extended with the concepts that are described in this Chapter. This reasoner is in fact an explainable deductive FOPL reasoner and provides Toulmin Models containing information concerning its reasoning. In the remaining of this section, this reasoner shall be referred to as Simple First Order Predicate Logic Reasoner (SFOPLR). SFOPLR has been developed purely for demonstrative purposes and is therefore not complete and (probably) not sound either, but since its purpose is solely to explain and clarify its reasoning, the validity of the reasoning itself is not of importance.

SFOPLR can be fed with FOPL rules and facts. Subsequently, it can be questioned and returns whether a formula can be deduced from the given rules and facts. SFOPLR applies simple rules like *modus ponens* and can also deduce from earlier deduced facts or rules, like. It also attempts to deduce using the exclusion of premises in formulae with a logical or. The following formulae are examples of what SFOPLR is able to deduce:

$$\frac{p \rightarrow (p' \rightarrow q), p, p'}{q}$$

$$\frac{p \vee q, \neg p}{q}$$

This reasoner has been adapted to an explainable agent as described in this Chapter. It generates Toulmin Models with FOPL for each transition and stores these in a ToulminBase. A prototype of Clara has been implemented that takes this ToulminBase and presents an explanation. Figure 4.9 shows the output of SFOPLR explained by this prototype of Clara and after clicking on 'john has a fever', 'john is a mammal' and 'john is a man'. SFOPLR was given the following rules and subsequently queried whether it holds that $Has(john, influenza)$:

Algorithm 3 Feeding SFOPLR With Rules and Facts

```
1: model.add(Formula.parseFormula("FORALL x : Mammal(x) & Has(x, a_headache) & Has(x, a_fever) & Has(x, a_sore_throat) ->Has(x, influenza)"));
2: model.add(Formula.parseFormula("FORALL x : Man(x) ->Mammal(x)"));
3: model.add(Formula.parseFormula("FORALL x : Human(x) & ~Woman(x) ->Man(x)"));
4: model.add(Formula.parseFormula("FORALL x : Human(x) & (temperature(x) >37) ->Has(x, a_fever)"));
5: model.add(Formula.parseFormula("Human(john)"));
6: model.add(Formula.parseFormula("temperature(john) >37"));
7: model.add(Formula.parseFormula("~Woman(john)"));
8: model.add(Formula.parseFormula("Has(john, a_headache)"));
9: model.add(Formula.parseFormula("Has(john, a_fever)"));
10: model.add(Formula.parseFormula("Has(john, a_sore_throat)"));
```

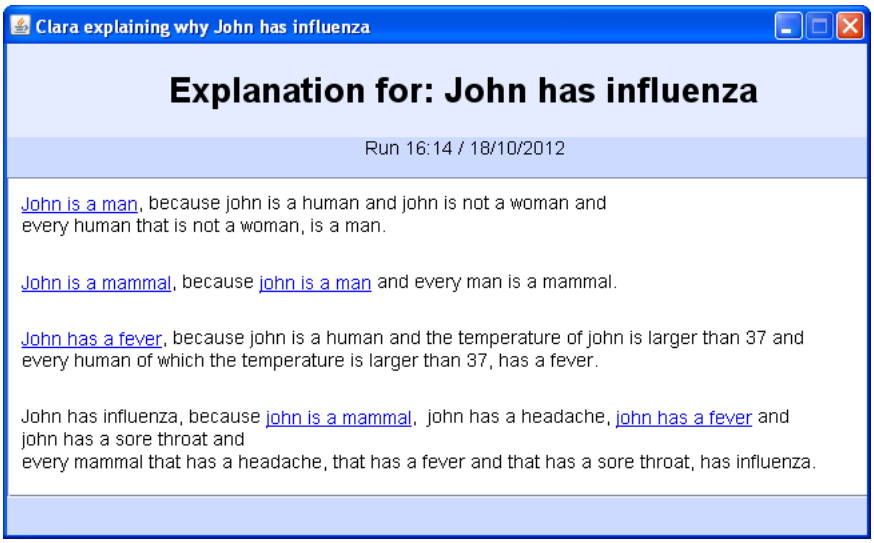

Figure 4.9: Example Explanation for FOPLR

4.7 Chapter Conclusion

This Chapter proposed a design for the implementation of the automatically generation of explanations for HeMAS. It has stated that HeMAS agents need to be classified as, what is called in this thesis, a Limpid or Cloudy agent. The difference is that Limpid agents can be explained in more detail, while for Cloudy agents a black-boxed approach is pursued.

It was stated that the Toulmin Model can be used as an envelope containing the information delivered by the explainable agents. Concatenating several Toulmin Models to Toulmin Chains allows the realisation of the desired click-through mechanism. The feasibility of Toulmin Chains has been shown by the proposal of an actual Java implementation.

Apart from a model for storing the information obtained from the explainable agents, the need for an interlingua for this information has been pointed out. FOPL has been chosen as the form in which the explainable agents should offer their information to Clara. It has been shown how FOPL can be implemented in an object oriented programming language. It was also shown that in order to convert FOPL to NL, a lexicon is required. WordNet by means of JAWS seemed appropriate for this.

Also, it was stated that the conversion of FOPL to NL could be done using external templates for a better maintainability and expandability. For the realisation of this, the use of LogicBabelfish has been suggested.

The next Chapter will unfold LogicBabelfish and its characteristics further and hopefully convince the reader of its usefulness within the present context.

Chapter 5

LogicBabelfish

LogicBabelfish is a small programming language, dedicated to the translation of formal language to natural language. In other words: it is an NLG tool. The name was derived from the fictive creature Babelfish, which made its occurrence in Douglas Adam's novel The Hitchhiker's Guide to the Galaxy. This little fish-formed creature would 'swim' into one's ear to translate any alien language to the native language of the owner of the ear. LogicBabelfish is intended also to translate to any (western) NL.

This Chapter describes the LogicBabelfish programming language by means of its syntactic and semantic properties with Section 5.2 and 5.3 respectively. The third Section gives an indication of how it could be used in practice. First, a short introduction to LogicBabelfish is given in the next Section.

5.1 Introduction

LogicBabelfish is a declarative programming language that can be used to translate sentences of a logical formalism to a different form, like NL. The functioning is comparable to an SQL database; once the base is built, it allows for consultation with queries and returns results from what can be deduced from the base. More specifically, one programs translation patterns for the input Logical Form in the LogicBabelfish program. Then on querying, it tries to match the query with these patterns, applies the translations of those that are matching and subsequently returns these translations. These principles are illustrated in figure 5.1.

There are no restrictions or obligations to the input and output of LogicBabelfish, meaning that a LogicBabelfish programmer is free to choose a logical formalism and the form of its translations. That said, this programmer will be needed to define when two sentences in his chosen formalism are matching. This freedom of choice allows LogicBabelfish to be applied in a wider scope. In this thesis though, the use of FOPL as input and English as output is assumed and shall also be used in the examples in this Chapter.

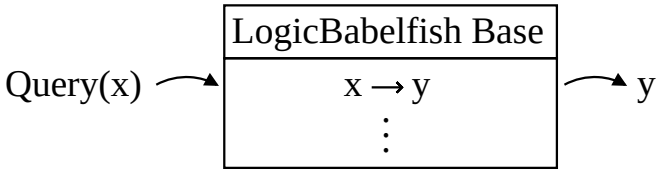

Figure 5.1: LogicBabelfish Illustration

5.1.1 LogicBabelfish Applications

A LogicBabelfish application consists solely of the translation patterns the previous Section mentioned, and possibly an external application that contains functions for string manipulations. Such a translation pattern is called a *Definition* and is used to translate a sentence of a query, matching the pattern of this Definition, to a different by the programmer specified form. The external application is nothing more than a collection of functions written in an imperative programming language that allow string manipulations or other small operations. These external functions should be written in the Java language, manifested in an executable Jar. Java has been chosen for the following reasons:

Platorm Independence With Java, LogicBabelfish retains its own platform independence, allowing a wider applicability.

Library Potential Java offers a large library with many functionalities, including many string operations. These may prove quite useful in the context of NLG.

Popularity Java currently is a popular programming language and has been used for many applications[1]. By embracing the same language, these applications can easily be included and benefited from.

5.1.2 LogicBabelfish as NLG Tool

If LogicBabelfish were to be placed in the architecture of traditional NLG systems as shown in figure 2.4 on page 24, it would float somewhere above the Micro Planner and Surface Realiser. LogicBabelfish does Micro Planning, because it can be used to make choices for expressions with words. It is a Surface Realiser as well, since it transmutes a text specification to surface sentences in NL. To be more precise, I like to make the distinction between LogicBabelfish sourcecode and the compilation and Interpretation process, and place these phenomena in the NLG architecture seperately. The programming in LogicBabelfish would then be the Micro Planning and the compiling and interpreting would be the Surface Realisation.

[1]The previous Chapter already mentioned WordNet as an example.

5.2 Syntax

This Section covers the syntax of LogicBabelfish.

Definition 1 (*LogicBabelfish Application*) A LogicBabelfish application is denoted by the tuple $\langle \nu, \Delta, \epsilon \rangle$, where ν is the name of the LogicBabelfish base, Δ is the set of all Definitions $\{\delta_1, ..., \delta_n\}$ and ϵ is a reference to external functions.

A LogicBabelfish application can be subdivided into several files, characterised by the '.lb' extension. The compilation and linking of different LogicBabelfish files is the responsibility of the LogicBabelfish compiler. This compiler takes a directory and compiles and links all contained LogicBabelfish files.

5.2.1 Definition

Definitions are the core of LogicBabelfish, they define how certain forms of the formalism in use should be interpreted as text. They look somewhat like *functions* in C and C^{++} and such languages, as they can be 'called' and have some sort of parameters and return a value. In contrast with functions though, Definitions take (static) identifiers and a logical formula, the return types are always strings and multiple strings can be returned in one Definition. A Definition consists of four parts:

1. **Name** The name can be used to call a specific Definition. By convention, the name of a Definition should start with a lowercase letter, or with an underscore. Names do not necessarily need to be unique, as one can overload Definitions.

2. **Logical Form** This defines the form of a formula that can be interpreted by the regarding Definition. When a Definition is called, a formula is given to ground the Logical Form. A formula consists of what in this context are called *Formparts* and operators to bind Formparts. So taking FOPL as formalism, the formula $x \wedge y$ consists of the Formparts x and y and an infix operator $\wedge$. Since LogicBabelfish is unprejudiced about the formalism in use, it cannot be responsible for parsing the Logical Form and thus should be done externally.

3. **identifier(s)** Definitions can also have zero or more identifiers. They can be used to identify what a Definition does, like for example which NL it returns. It is forbidden to have multiple identifiers with the same name, as well as identifiers with the same name as a Formpart.

4. **Interpretation(s)** An Interpretation does what it assumes; interpret a Logical Form and return a string. A Definition can have one or more interpretations, causing one or more possible translations. During runtime, all interpretations in the call sequence are visited and a list of possible translations for the given query is returned.

Definition 2 (*Definitions*) A LogicBabelfish Definition δ is represented as the tuple $\langle d, f, I, P \rangle$, where d is the name of the Definition, f is the Logical Form, I is a set of identifiers $\{id_1, ..., id_n\}$ and P is a set of Interpretations $\{p_1, ..., p_m\}$.

Definition 3 (*Interpretation*) An Interpretation $p_i \in P$ is a sequence of Interpretationparts $(p_{i_1}, ..., p_{i_n})$, where p_{i_j} can be a call, string, variable or Formpart.

Example 1:

The code below shows what a Definition may look like. Here the name (d) is $myDef$, the Logical Form (f) is $P(t)$, the sequence of identifiers (I) is (EN) and the set of interpretations (P) is $\{(t + \text{“ is a ”} + P)\}$.

define $myDef[P(t)](EN)$
  $t +$ “ `is a` ” $+ P$ **.**

In the Section about the semantics, the Definition is revisited and explained in more detail.

5.2.2 Reference to External Functions

A reference to external functions in a Jar file, is done in one of the LogicBabelfish files. It should hold a relative path with respect to the location of the LogicBabelfish file that contains the reference, or an absolute path to the Jar file.

Example 2:

A reference to the external functions can be done as follows:

use “`../../MyJar.jar`” **.**

5.2.3 EBNF

The LogicBabelfish syntax in Extended Backus Naur Form is defined as follows:

character = *digit* | *letter* | "_" ;
definition = "`define`" , *whitespace* , *definitionHeader* , *whitespace* , *interpretations* , [*whitespace*] , "`.`" ;
definitionCall = "`@`" , *word* | "`?`" , "`[`", [*whitespace*], *logicalForm* , [*whitespace*] , "`](`", [*whitespace*], [*identifiers*], [*whitespace*], "`)`" ;
definitionHeader = *definitionName* , "`[`" , [*whitespace*], *logicalForm* , [*whitespace*] , "`]`" , [*whitespace*] , "`(`" , [*identifiers*] , [*whitespace*] , "`)`" ;
definitionName = *word* ;
digit = "`0`" | "`1`" | ... | "`8`" | "`9`" ;
functionCall = *functionName* , "(", [*whitespace*] , *logicalForm* | *identifier* | *string* | *functionCall* , [*whitespace*] , ")" ;
functionName = *word* ;
identifier = *word* ;
identifiers = [*whitespace*] , *identifier* |*definitionCall*, [*whitespace*] , ["`,`" , *identifiers*] ;
inclusion = "use" , *whitespace* , *pathToJar* ;
interpretation = *interpretationPart* | *interpretation* , [*whitespace*] , "`+`" , [*whitespace*] , *interpretationPart* ;
interpretationPart = *word* | *definitionCall* | *functionCall* | *string* | *variable* ;
interpretations = *interpretation* | *interpretations* , [*whitespace*] , "`;`" , *interpretation* ;
letter = *uletter* | *lletter* ;
lletter = "`a`" | "`b`" | ... | "`y`" | "`z`" ;
logicalForm = "`[`" , (? `Well-Formed Sentence in Formalism` ? | *variable*) , "`]`" ;
pathToJar = *string* ;
string = '`"`' , *word* , '`"`' ;
uletter = "`A`" | "`B`" | ... | "`Y`" | "`Z`" ;
variable = "`$`" , *word* ;
whitespace = *whitespacePart* | *whitespace* , *whitespacePart* ;
whitespacePart = " " | \t | \n ;
word = *character*+
whitespacePart = " " | \t | \n ;
whitespace = *whitespacePart* | *whitespace* , *whitespacePart* ;

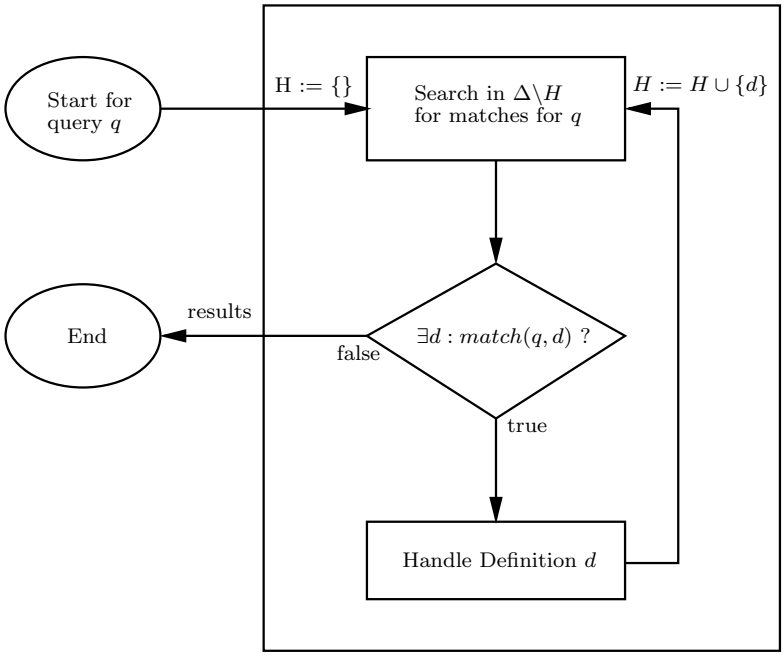

Figure 5.2: LogicBabelfish Handle Query

5.3 Semantics

This Section covers the semantics of LogicBabelfish. Figure 5.2 illustrates how LogicBabelfish interprets a query. On querying LogicBabelfish, a matching Definition is searched for in Δ. If one is found, the Definition is handled - this is illustrated in figure 5.3 - and excluded from future searches for matches for the current query. If no more matching Definition can be found for a query, a set of all resulting translations is returned (which may be empty).

Figure 5.3 illustrates how the LogicBabelfish interpreter handles a Definition. First each Formpart in the Logical Form is grounded by the matching elements of the formula in the query. Subsequently, all Interpretations are handled (see figure 5.4) and their results are returned.

Figure 5.4 is to illustrate how LogicBabelfish Interpretations are handled. First, the most left Interpretationpart is selected (which is p_1 initially). It is then determined what type the selected Interpretationpart is. If it is a Definition call, this call is handled correspondingly, resulting in the same mechanism that is shown in figure 5.2 and the resulting string is concatenated to the result of the current Interpretation. Similarly, a Function call is executed correspondingly and this result is also concatenated to the resulting string of the current Interpretation. In all other cases, (that is, if the Interpretationpart is a string,

Formpart or variable), the value of the Interpretationpart is simply concatenated to the result without extra effort. Subsequently, after concatenation, it is checked whether the just handled Interpretationpart was the final one of the sequence. If so, the result of this Interpretation is returned. Otherwise, the next Interpretationpart is selected and the whole procedure is repeated.

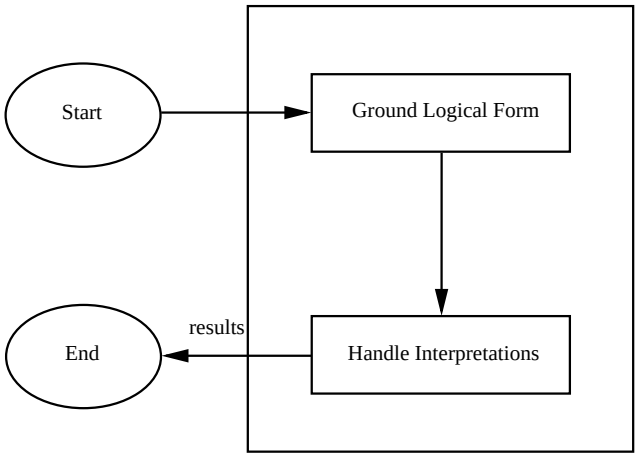

Figure 5.3: LogicBabelfish Handle Definition

5.3.1 Grounding

In LogicBabelfish, grounding was chosen as the term to refer to assigning a formula to the Logical Form of a Definition, comparable (but not identical) to grounding in Prolog. When a Logical Form is grounded, all appearances of Formparts in the Definition are grounded as well, similar to the value assignment of parameters in classical programming languages like C. A Logical Form can only be grounded by a matching formula. As was already mentioned in the introduction of this Chapter, the groundability is not included in the LogicBabelfish interpreter, but should be defined independently. The LogicBabelfish interpreter must be able to 'ask' the independent parser whether two formulae are matching. When a match is found, this parser must also determine the values per Formpart.

Example 3:

The following example illustrates the grounding of a Logical Form and its Formparts:

define _[$P(t1, t2)$]()
  $t1$ + " `is a` " + P + " `of` " + $t2$ `.`

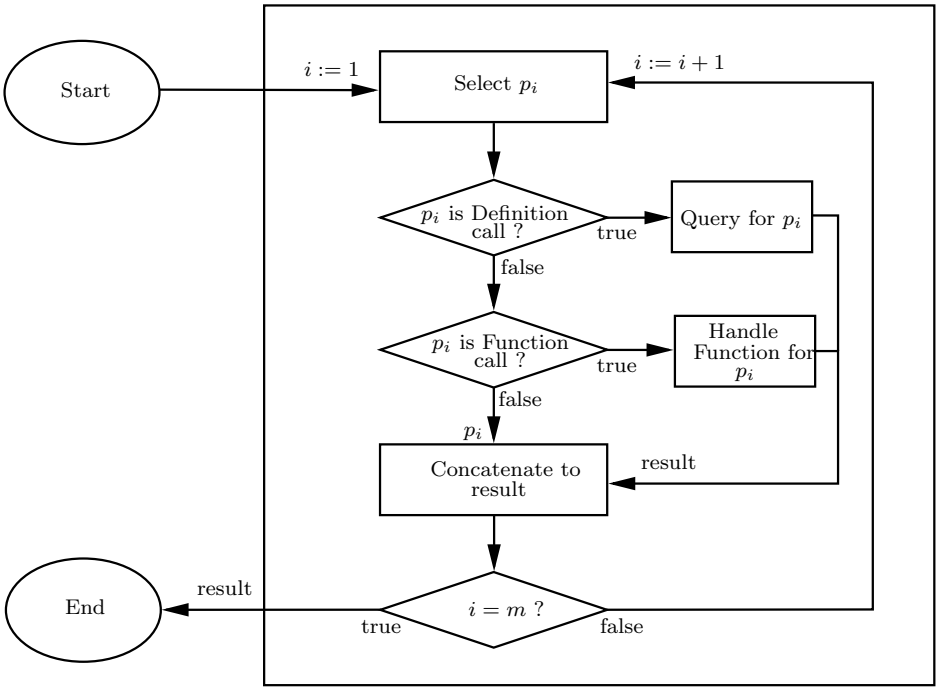

Figure 5.4: LogicBabelfish Handle Interpretation

This Logical Form matches the formula $Friend(john, jack)$, as both the Logical Form and the formula are atomics and both take two arguments. Thus this Logical Form can be grounded with the regarding formula and the interpretation would be 'john is a Friend of jack'.

Variables

Variables are special Formparts that tolerate any well-formed formula. They can be a part of the Logical Form like any other type of Formpart, or they could *be* the Logical Form. If this is the case, then every formula will match the Formpart.

Example 4:

The code below shows a Definition with a variable as Logical Form, making this Definition applicable for any well-formed formula. As Section 5.3.4 about the priorities of Definitions will defend, this allows one to create a Definition providing an error message if no suiting translation could be found for the given formula.

define _[$*v*]()
"No match found for " + $v .

5.3.2 Definition Calls

Definition calls can be identified by the at sign '@', followed by either the name of a Definition or a question mark. A call with the latter is named an *anonymous call*, since the programmer does not explicitly - though implicitly - make clear which Definition is to be called. Calling a Definition with a Logical Form causes the Interpretation(s) of the called Definition to be applied to the given Logical Form and to return it to replace the call statement with this result.

Example 5:

In this example, a Definition has one Interpretation which starts with a Definition call.

define _[$*v*]()
@*anotherDef*[$*v*]() .

All regular calls could be replaced by anonymous calls provided with sufficient identifiers and corresponding Definitions. Yet, such constructions may limit the readibility of a LogicBabelfish application. It is much clearer to directly refer to the actual name of a Definition, rather than using anonymity. It is therefore adviced to always use non-anonymous calls if possible, at least for constructions such as lexicons, bridges, and such.

Definition 4 (*Call*) A LogicBabelfish call c is represented as the tuple $\langle d, f, I\rangle$, corresponding to the name, Logical Form and identifiers of the Definition that is called.

A Definition can be called if the call 'matches' the regarding Definition. Let $match(c, \delta)$ be a function that returns Definition δ if and only if call c matches δ:

$$match(\langle d, f, I\rangle, \langle d', f', I', P\rangle) \quad \stackrel{\text{def}}{=} \quad (d =? \vee d = d') \wedge f \;\stackrel{\texttt{GND}}{=\!\!\leftarrow}\; f' \wedge I = I'$$

The expression $d =?$ is to determine whether the name of the call is equal to the question mark, or rather, whether the call is anonymous. The expression $f \;\stackrel{\texttt{GND}}{=\!\!\leftarrow}\; f'$ denotes that f is groundable by f' as a result of the logical equivalence of f and f'. Again, this expression is not to be defined by LogicBabelfish.

Grounding in calls

In a call, everything is 'groundable', except the name of the Definition that is to be called.

Example 6:

Consider the following example Definition:

define _[$P(t)$]()
  @P[$P(t)$](P) .

If the Logical Form of the above Definition is grounded with the formula Man(john), the call during runtime would then be @$P[Man(john)](Man)$. All groundable parts of the call - the Formpart and identifiers - are grounded, but the name of the Definition in the call remains untouched. The reason behind this semantical issue is readibility and predictability. If the directions of calls would be dynamic - which they would be if the name of calls would be groundable - the behaviour of a run of an application could appear arbitrarily. Furthermore, I believe that the mixture of the programming structure (the call name) with runtime behaviour or the actual use of the application (the grounded Logical Form) should be avoided.

5.3.3 Cyclic Dependency

In LogicBabelfish, a Definition is dependent on another Definition if it contains an Interpretation that has a call pointing to that other Definition.

Definition 5 (*Dependency*) Let C denote the set of all calls in a LogicBabelfish application, we have:

$$dep(\langle d, f, I, P\rangle, \delta) \quad \stackrel{\text{def}}{=} \quad \exists i, j : p_{i_j} \in C \wedge match(p_{i_j}, \delta)$$

When in LogicBabelfish Definition δ is dependent on Definition δ', we can also state that there exists at least one *Callpath* from δ to δ'. A Callpath does not necessarily involve dependency directly though, as the following Definition will show.

Definition 6 (*Callpath*) The Callpath from δ to δ' is defined as follows:

$$callpath(\delta, \delta') \quad \stackrel{\text{def}}{=} \quad \{dep(\delta, \delta_1), dep(\delta_1, \delta_2), ..., dep(\delta_i, \delta')\}$$

Cyclic Dependency occurs when there exists a Callpath from a Definition pointing to itself: $callpath(\delta, \delta)$. The acceptability for Cyclic Dependency is partly

dependent on what sort of it occurs. LogicBabelfish makes the distinction between Explicit Cyclic Dependency and Implicit Cyclic Dependency. The former is recognisable by the compiler, because the Callpath has been made explicitly known by defining the names and identifiers in the calls and by the use of Logical Forms that do not contain variables. The compiler should demand removal of explicit recursive constructions, because since all calls are static, there is no stop-condition and the program will loop through the calls ad infinitum. Implicit Cyclic Dependency on the other hand, is not noticeable by the compiler. It occurs when during runtime a Definition is called that has a Formpart with one or more variables. Implicit Cyclic Dependency is not per se invalid.

Recursion

Recursion is in fact a special case of Cyclic Dependency. It can be denoted by $|callpath(\delta, \delta)| = 1$ or even simpler by $dep(\delta, \delta)$.

Example 7:

This example will show a LogicBabelfish run according to Figure 5.2, 5.3 and 5.4 with Implicit Recursion. Two Definitions are defined here:

define $a[\$v1 \wedge \$v2]()$
  @?[$\$v1$]() + " `and` " + @?[$\$v2$]() .
define $b[P(t)]()$
  t + " `is a` " + P .

As these are the only Definitions in this example, Δ is defined as follows:

$$\Delta = \{\langle a, \$v1 \wedge \$v2, \emptyset, (\langle ?, \$v1, \emptyset\rangle, \text{" } \texttt{and} \text{ "}, \langle ?, \$v2, \emptyset\rangle)\rangle, \langle b, P(t), \emptyset, (t, \text{" } \texttt{is a} \text{ "}, P)\rangle\}$$

The following query shall be used in this example:

$$q = \langle ?, (Man(john) \wedge Woman(mary)) \wedge Cat(fluffy), \emptyset\rangle$$

The run of this example can be described as follows:

Run:

1. Search in $\Delta \backslash \emptyset$ for matches for q
2. $match(q, \langle a, \$v1 \wedge \$v2, \emptyset, (\langle ?, \$v1, \emptyset\rangle, \text{" } \texttt{and} \text{ "}, \langle ?, \$v2, \emptyset\rangle)\rangle)$
3. $(Man(john) \wedge Woman(mary)) \overset{\text{GND}}{\rightarrow} \$v1, Cat(fluffy) \overset{\text{GND}}{\rightarrow} \$v2$
4. Handle $q' = \langle ?, Man(john) \wedge Woman(mary), \emptyset\rangle$

5. Search in $\Delta \backslash \emptyset$ for matches for q'
6. $match(q', \langle a, \$v1 \wedge \$v2, \emptyset, (\langle ?, \$v1, \emptyset \rangle, \text{“ } \texttt{and} \text{ ”}, \langle ?, \$v2, \emptyset \rangle) \rangle)$
7. $Man(john) \overset{\texttt{GND}}{\rightarrow} \$v1, Woman(mary) \overset{\texttt{GND}}{\rightarrow} \$v2$
8. Handle $q'' = \langle ?, Man(john), \emptyset \rangle$
9. Search in $\Delta \backslash \emptyset$ for matches for q''
10. $match(q'', \langle b, P(t), \emptyset, (t, \text{“ } \texttt{is a} \text{ ”}, P) \rangle)$
11. $Man \overset{\texttt{GND}}{\rightarrow} P, john \overset{\texttt{GND}}{\rightarrow} t$
12. Handle $john$
13. Handle “ `is a` ”
14. Handle Man
15. Handle “ and ”
16. Handle $q''' = \langle ?, Woman(mary), \emptyset \rangle$
17. Search in $\Delta \backslash \emptyset$ for matches for q'''
18. $match(q''', \langle b, P(t), \emptyset, (t, \text{“ } \texttt{is a} \text{ ”}, P) \rangle)$
19. $Woman \overset{\texttt{GND}}{\rightarrow} P, mary \overset{\texttt{GND}}{\rightarrow} t$
20. Handle $mary$
21. Handle “ `is a` ”
22. Handle $Woman$
23. Handle “ `and` ”
24. Handle $q^* = \langle ?, Cat(fluffy), \emptyset \rangle$
25. Search in $\Delta \backslash \emptyset$ for matches for q^*
26. $match(q^*, \langle b, P(t), \emptyset, (t, \text{“ } \texttt{is a} \text{ ”}, P) \rangle)$
27. $Cat \overset{\texttt{GND}}{\rightarrow} P, fluffy \overset{\texttt{GND}}{\rightarrow} t$
28. Handle $fluffy$
29. Handle “ `is a` ”
30. Handle Cat

Result: "`john is a Man and mary is a Woman and fluffy is a Cat`"

The query matches Definition a (2), thus Definition a is selected and its Logical Form is grounded (3). Next, the first Interpretationpart, which is a Definition call, is handled (4) resulting in another search for a match (5). This new query also matches Definition a (6), so similarly, the Logical Form is grounded (7) and the first Interpretationpart is handled (8). After another search (9), Definition b is found to be matching (10), so its Logical form is grounded (11) and its Interpretationparts are handled sequentially (12, 13, 14). Then, the interpreter jumps back to an earlier execution of a of which the next Interpretationpart is selected and handled (15). This procedure is continued in the same way until no more Interpretationparts are to be handled and the final result is returned, which is the sentence "john is a Man and mary is a Woman and fluffy is a Cat".

5.3.4 Priorities

It has been stated earlier in this Chapter that when a Definition is called, all available interpretations are used to return a set of possible translations. But it is also possible - and for large applications probable - that more than one Definition match a query. This Section will give the exact rules for what LogicBabelfish does when multiple possible Definitions are encountered.

Let $matchingDefs(c)$ represent a function returning a set of all Definitions that match call c:

$$matchingDefs(c) \quad \stackrel{\text{def}}{=} \quad \{\delta \in \Delta \big| match(c, \delta)\}$$

Definition 7 (*Priority*) Let $query(c, \langle \nu, \Delta, \epsilon \rangle)$ represent the querying of a LogicBabelfish application $\langle \nu, \Delta, \epsilon \rangle$ with call c, returning a set of translations as string. Let $vars(f)$ denote a function that returns a set of all variables that are a Formpart of f and let $prio(\delta, \delta', c)$ denote that δ has priority over δ' for call c:

$$prio(\langle d, f, I, P \rangle, \langle d', f', I', P' \rangle, c)$$
$$\stackrel{\text{def}}{=}$$
$$\{\langle d, f, I, P \rangle, \langle d', f', I', P' \rangle\} \subseteq matchingDefs(c) \wedge ($$
$$(\big|vars(f)\big| = 0 \wedge \big|vars(f')\big| > 0) \vee$$
$$(\big|(vars(f')\big| > 0 \wedge \big|vars(f)\big| > \big|vars(f')\big|))$$

So δ has priority over δ' if both Definitions are candidates for call c (otherwise, choosing between these Definitions would not be an issue) and either δ has no variables in its Logical Form and δ' does, or both δ and δ' have variables in their Logical Form, but δ's has more.

The idea of this prioritising mechanism is that Definitions that are more specific deserve a higher priority. A Definition with no variables is, so to speak, non-tolerant; only formulae with exactly the same abstract form can apply to it. Apparantly the programmer had exactly in mind how the Logical Form

of such a Definition has to be interpreted. Similarly, a Definition with many variables is more specific than a Definition with a few or only one variable, because more information about the Logical Form is given; a Logical Form with many variables must have many logical operators to bind these variables.

It is though possible to work around these priorities if necessary. There might be cases in which a programmer wants to have a tolerant Definition be called before a less-tolerant Definition is touched or maybe at the 'same' moment. The first situation can be reached by using identifiers in the Definition and adjusting the Definition calls to match the identifiers of the tolerant Definition. the second situation can be achieved by adding an Interpretation with an explicit Definition call to the Definition with the higher priority. In Section 5.4.5, an example is given of why one could want this and how it is done.

5.3.5 Interpretations

Interpretations can be regarded as the 'return-statements' of the Definitions. They have the form of string concatenations in Java and a treated likewise; the parsing starts at the left and moves to the right. When a call is encountered, this call is executed before the rest of the Interpretationparts are parsed. Identifiers are just converted to strings and are concatenated to the resulting string. Formparts have been grounded and thus concatenate their new value as string, this also applies to variables. An Interpretationpart could also be a string, in which case it is literally concatenated to the result. Interpretations are completely independent of each other, meaning that if a Definition has several Interpretations, they will never affect one another.

5.4 General Appliance

This Section will provide practical information of how LogicBabelfish can be applied.

5.4.1 Default Identifiers

Although the LogicBabelfish programmer is free to choose his own identifiers, there are some properties that often need to be identified. The following list applies to Definitions that are directly responsible for the near-surface text:

- **Language** Obviously, it is necessary to identify the language to which the Definition should translate.
- **Word types** Identifying the word types is critical [2], because it directly influences how the formula should be translated. Word types include terms like 'Noun', 'Verb', 'Adjective'.

[2] See Chapter 4.5.3.

- **Tense** Identifiers like 'Past', 'Present' or 'Future' represent the tense of a Definition.
- **Active/Passive form** If one wants to be able to distinct between active and passive sentences, an identifier like this is needed.
- **Plurality** For determining whether the formula is in plural or singular form.

It remains language-dependent which identifiers are required. The above list was composed with the English language in mind.

Example 8:

The following example shows a Definition with the suggested identifiers:

define $atomic[P(t)](En, Noun, Future, Active, Single)$
t + " `will be a ` " + P **.**

Remember that Definitions need to be called with the specific identifiers. So somehow, the identification of for example word types, needs to be done before the calling of the Definition. This identification could be done in a program before querying LogicBabelfish, but Section 5.4.3 shows how this can be done withing LogicBabelfish.

5.4.2 Logical Operators

Still assuming FOPL is the formalism in use, it is required to create a Definition for every logical operator. Probably the best way to do this is using tolerant Logical Forms, so that formulae with multiple operators can be recursively parsed.

Example 9:

The following Definition can be used to translate logical sentences containing a logical and:

define $logicalAnd[\$v1 \wedge \$v2]()$
@?[$\$v1$]() + " `and ` " + **@**?[$\$v2$]() **.**

Section 5.3.3 of this Chapter showed how constructions like this are interpreted and what result is returned. All logical operators require similar constructions, with an exception for quantifiers.

Quantifiers

Quantifiers are somewhat harder to translate if we want their NL form to be as natural as humans are used to translate them. For example, a non-natural, but straightforward translation for the formula $\forall x : Man(x) \rightarrow Mortal(x)$ would be 'For all x it is the case that if x is a man, then x is mortal". A much more appealing translation would be something like 'All men are mortal" or 'Every man is mortal". The first translation would be easy to implement in LogicBabelfish, since it requires just the concatenation of the quantifier-part 'For all x it is the case that" before the translation of a logical operator. the second translation is much harder, because it requires the conjugation of the verb 'to be' to make it 'are' and it must somehow obtain the plural form of 'man'. The third translation seems to me elegant enough, is easier to implement and therefore a fair translation. The following example will show how one can achieve translations of this form.

Example 10:

define $univQtfr[\forall x : P1(x1) \rightarrow P2(x2)]()$
  "`every ` " + $P1$ + @$univQtfr[P2(x2)]()$ **.**
define $univQtfr[P(x)]()$
  " `is a ` " + P **.**

Although the Definitions in the example will be able to translate the formula $\forall x : Man(x) \rightarrow Mortal(x)$ correctly, they do not cover most other formulae. An observant reader might have noted that the second Definition has been focussed on *adjectives*. A formula like $\forall x : Man(x) \rightarrow Person(x)$ will not be translated correctly. This must be corrected by identifying word types and implementing a lexicon is a way to do it.

5.4.3 Implementing Lexicons

It is possible to build lexicons in LogicBabelfish. By doing so, one could integrate the Micro Planning and the Surface Realisation of NLG into a single application. In Chapter 4.5.3, it was explained why a NLG system requires a lexicon: primarily to identify the word types of the parts of the formulae that are to be translated. So a lexicon does not have to be more than just a lookup-table for words. This can be achieved in LogicBabelfish using identifiers.

Example 11:

The following example illustrates how a simple Definition can be implemented to return the type of a specific word:

define $lexicon[P(t)](Man)$
  @?$[P(t)](Noun)$ **.**

Calling this Definition should be done like **@***lexicon*[$P(t)$]**(***P***)**, where the grounding of P is given as identifier in the call. The lexical Definition then calls for another Definition with the required new information added: the word type of P. Subsequently, a Definition is required that expects corresponding identifiers and handles the final translation given the word type.'

Calling External Lexicons

An alternative to the implementation of a lexicon in LogicBabelfish, is the calling of an external lexicon. In Chapter 4.5.4, WordNet by means of JAWS was suggested as lexicon. Since JAWS is a Java application, it can be applied by LogicBabelfish without too much effort.

Example 12:

The following Definitions would adequately translate sentences with atomic predicates concatenated with a logical or. Here, an external call 'getWordType' is used to determine the word type of the predicate and this word type is passed as identifier in order for the Definition call to match the corresponding Definition.

define *logicalOr*[$\$v1 \vee \$v2$]**()**
  @?[$\$v1$]**(***getWordType*($\$v1$)**)** + " `or` " + **@**?[$\$v2$]**(***getWordType*($\$v2$)**)** **.**

define *atomic*[$P(t)$]**(***Noun***)**
  t + " `is a` " + P **.**
define *atomic*[$P(t)$]**(***Adjective***)**
  t + " `is` " + P **.**
define *atomic*[$P(t)$]**(***Verb***)**
  t + P **.**

5.4.4 Implementing Bridges

In logical formalisms, one often encounters abstractions. For example, the information that two persons love each other can be abstractly noted as $L(p1, p2)$. Such abstract forms cannot be translated to NL without 'bridging' between the abstraction and the actual meaning. That is, in this example, bridging from $L(p1, p2)$ to $Loves(person1, person2)$. This bridging can be easily implemented in LogicBabelfish.

The bridging Definitions are quite similar to lexical Definitions as in the previous Section. Bridges are characterised by the fact that they can provide multiple different translations for one abstract formula. LogicBabelfish shares this characteristic and is thus a well-suited tool for bridging. The example Definition below shows how bridging can be implemented.

Example 13:

In this example, the abstract predicate $P(t1, t2)$ is bridged to $Teacher(t1, t2)$ and $Student(t2, t1)$:

define $bridge[P(t1, t2)]()$
  @?$[Teacher(t1, t2)]()$;
  @?$[Student(t2, t1)]()$.

This Definition states that the abstract formula $P(t1, t2)$ should be interpretated as either $Teacher(t1, t2)$ or $Student(t2, t1)$. Calling a bridge is done identically to calling a lexical Definition, so in this case a correct call would be **@**$bridge[P(t1, t2)]()$. Bridges can be combined with lexicons, naturally. The information that the lexicon would normally provide, should then be added to the calls in the Interpretations of the bridge.

5.4.5 Surface Realisation

LogicBabelfish can be used for Surface Realisation; the finishing touch on the final textual output. This can be quite practicable for simple changes like 'uppercasing' the first letter of the sentence, lowercasing all other letters and placing a dot at the end. To do this, one should create a Definition that is always called *first* (has the highest priority). This can be achieved by convention: creating a Definition with a certain name of certain identifiers and ensuring that the query is pointed towards this specific Definition.

Example 14:

In this example, the Definition that is to be called first is called 'main' and performs the simple surface realisations that were mentioned; realising a sentence starting with an uppercase letter and ending with a dot. The convention here would be that all queries with respect to the LogicBabelfish base containing the Definition of this example, should explicitly call this Definition.

define $main[\$v]()$
  uppercaseFirst(lowercaseAll(**@**?$[\$v]()$)) + "." .

More complicated surface realisations include the proper use of commas for summations, and referations to persons and objects with 'he', 'she', 'it' instead of full names repeatedly. For example the sentence 'john is a man and mary is a woman and fluffy is a cat' could be written more nicely as 'john is a man, mary is a woman and fluffy a cat'. Or even further, by removing identical verbs that follow each other, one could make the sentence 'john is a man, mary a woman and fluffy a cat' (but this is also a matter of taste of course). This surface realisation is perhaps better done outside LogicBabelfish or maybe in a different dedicated LogicBabelfish base.

5.4.6 A step to Natural Language Understanding

Although LogicBabelfish was intended to be a tool for NLG, it might also serve as a step towards NLU. In Chapter 4.4.1, the use of FOPL for NLG was defended. Similarly, FOPL could be useful in NLU, but on the opposite side of the input/output. Logic is often being used in AI for reasoning and denoting knowledge or beliefs, so it is not hard to imagine that the translation of NL sentences to FOPL can be an element of NLU. LogicBabelfish could play a role in the translation of NL sentences to FOPL, although the NL would be restricted to more rules than its natural grammar rules.

To use LogicBabelfish for NLU, one must invent a syntax for a NL and use it as the logical formalism of the LogigBabelfish application. It should also be defined, like for any logical formalism, when two abstract formulae - or sentences in this context - match each other. When this is done correctly, it is possible to implement Definitions the same way as in the rest of this Chapter. Suppose that we want to translate English to FOPL and we define the syntax of our formalism such that a sentence is placed between double quotes, where variables are placed in between outside the quotes. Then since variables always match anything, a Logical Form will match a sentence if the parts between the quotes are identical and if the variables can be grounded. Hopefully, the example below makes this clear enough.

Example 15:

The following Definition could be used for a simple form of NLU, provided a corresponding Logical Form matcher has been implemented.

```
define _[$v1" is a "$v2]()
    uppercaseFirst($v2) + "(" + $v2 + ")" .
```

This Definition would match a sentence like "john is a man" and return "Man(john)". Similar Definitions could be implemented for other constructions comparable to the logical operators.

The use of LogicBabelfish for NLU has not been studied in detail (yet), as it falls outside the scope of this thesis.

5.5 Chapter Conclusion

In this Chapter, the small programming language LogicBabelfish has been presented as a tool to translate sentences of a logical formalism to a different form, particularly NL. LogicBabelfish is a declarative programming language that can be used to build a base of Definitions that dynamically apply the translation.

It is believed that LogicBabelfish offers sufficient mechanisms for a satisfying conversion of FOPL to NL. Future work on LogicBabelfish should first of all include its actual implementation as this has only been done partially as of yet. Other future work could include research to the use of LogicBabelfish for NLU.

LogicBabelfish is believed to make a useful NLG tool and a fitting final piece in the puzzle towards a solution for the second research question.

Chapter 6

Conclusion

This thesis has described the research that was conducted towards automatically explaining the behaviour of AI systems in context of HeMAS, a multi agent system applying different AI and datamining techniques. As such, the conducted research is one of Explainable AI. It is also research in the field of NLG, since the explanations are to be textual explanations. The goal has been stated as follows:

To automatically explain the different datamining and AI techniques within HeMAS, such that the ultimate result of the execution of these techniques is more transparent, leading to an improvement of HeMAS' trustworthiness.

In this Chapter, an evaluation of the achievement of this goal is provided. This is done by addressing the research questions again and giving an outline of the contribution of this research. The next Section will be covering this, while the last Section provides suggestions for future related research.

6.1 Outline of Contribution

Improving the transparency of HeMAS inevitably involves explaining each AI technique that is included in the agents of this multi agent system. Hence the first research question was formulated accordingly as follows:

How can the behaviour of an AI technique be sufficiently explained?

It was found that there was no sufficient solution available, mainly due to the versatile character of the HeMAS agents [1]. The approach that has been proposed in this thesis, consists of two important parts: the implementation of the explanatory agent Clara and the adaption of current HeMAS agents to explainable HeMAS agents. These two parts should allow the automatically generation

[1] See Chapter 1.3.

of NL explanations presented in a click-through-mechanism, which in turn - as defended in this thesis [2] - should lead to a greater trust in HeMAS. The explanations are to be clear and should not only trace an explainable agent's behaviour, but also justify it.

The feasibility of the proposed solution to the first research question, can be measured by answering the second research question concerning the actual implementation of the proposal. This second research question was defined as follows:

How can the automatically generation of explanations be realised in a generic way?

This question has been addressed by the fourth and fifth Chapter. In the former of the two, the necessity to discriminate the agents on the traceability of their AI techniques has been described and agents were to be classified Limpid or Cloudy with a different approach for each. The Limpid agents leave more space for the generation of explanations. It was found that the Toulmin Model could serve as an envelope for the information the explainable agents provide. It was also encountered that this provided information should have one single form, which was referred to as an interlingua. FOPL has been chosen as interlingua, which, in combination with WordNet and LogicBabelfish, is fairly translatable to NL.

6.1.1 Evaluation of the Hypothesis

Now that the goal and research questions have been treated, the hypothesis can be evaluated. The hypothesis was defined as follows:

The design decisions of the solution in this thesis are feasible and lead to an improvement of the trustworthiness of HeMAS.

The feasibility of the in this thesis proposed solution is tested by Chapter 4. The regarding Chapter has presented details on the implementation of the solution. It is believed that the use of the Toulmin Model and FOPL for the explainable agents is realisable for agent developers. These matters suggest a positive feasibility in accordance with the hypothesis. The most complete picture of this feasibility can be obtained by applying the solution to the actual HeMAS system, of course.

It is also believed that the solution causes an improvement of the user's trust in HeMAS. One of the fundamental parts of the solution, agent Clara, provides justifying explanations rather than solely traces of the explainable agent's reasoning. This is believed to be of importance [Swartout, 1983, Taylor et al., 2006]. The explanations are presented in such way that the user himself can decide the amount of information by simply clicking through the explanations, which is important according to [Pieters, 2011] and [Wallis and Shortliffe, 1982]. This short summary suggests corroboration of the hypothesis.

[2] See Chapter 2 and 3.

6.2 Future Work

There are several possible directions for future work based on the research described in this document. First, it should be noted that not all parts of the solution were actually implemented. The adaption of the actual agents to explainable agents and the implementation of all described functionalities of LogicBabelfish remain to be done.

One possible direction of future research could be towards a protocol for the justifications of Config Agents. This research should also include a choice of a data format and how this format can be translated to NL.

Another possible direction of future research could be the further exploration of LogicBabelfish. There may be matters that were initially not included in this programming language which could improve its usability or efficiency though. Also, the use of LogicBabelfish for NLU could be explored further. Other possible future research involving LogicBabelfish, could be smart selection of possible translations that are returned as a result from a query.

Another suggestion for further research is the further exploration of language independent NLG. There have been references to languages other than English in this thesis, but as was stated in the introduction, the goal has been reduced to generating the English language. Though some western languages can be quite difficult to generate natural sentences in, generating non-western languages might be an even greater challenge.

More research could also be conducted towards the automatically generation of figures along with the textual explanations. It was already mentioned that the use of the Toulmin Model can be a nice stepping stone to this. One could for example study the increase of users' comprehension due to the availability of figures alongside the text.

Finally, another direction of research could include further substantiation of the hypothesis by empirically studying the user's attitude towards HeMAS after the transparency improvements have been applied. I propose the selection of two groups A and B of a significant number of persons each. The persons of both groups are to be selected according to their background knowledge, which should be none or negligible in AI and equally in the specific medical field. All persons from both groups are presented an equal HeMAS configuration which is equally queried, with the only difference that in those configurations of group A Clara is included, while in those of group B she is excluded. After providing both groups ample time to consider the result, they are given a few questions concerning the trust in HeMAS. These questions should involve the person's comfort in the result and his attitude towards HeMAS.

Bibliography

[Adams, 1979] Adams, D. (1979). *The Hitchhiker's Guide to the Galaxy*. Random House Publishing Group, first edition.

[Anderson and Wright, 1988] Anderson, U. and Wright, W. F. (1988). Expertise and the explanation effect. *Organizational Behavior and Human Decision Processes*, 42(1988):250–269.

[Bench-Capon et al., 1991] Bench-Capon, T. J. M., Lowes, D., and McEnery, A. M. (1991). Argument-based explanation of logic programs. *Butterworth-Heinemann Ltd*, 4(3).

[Bratman, 1987] Bratman, M. (1987). Intentions, plans and practical reason. *Harvard University Press, Cambridge, MA*.

[Chandrasekaran et al., 1989] Chandrasekaran, B., Tanner, M. C., and Josephon, J. R. (1989). Explaining control strategies in problem solving. *IEEE Expert*, Spring(1989).

[Everett, 1994] Everett, A. M. (1994). An empirical investigation of the effect of variations in expert system explanation presentation on users' acquisition of expertise and perceptions of the system. Unpublished Doctoral Dissertation, University of Nebraska.

[Fellbaum, 1998] Fellbaum, C. (1998). A semantic network of english: The mother of all wordnets. *Computers and the Humanities*, 32(1998):209–220.

[Giboney et al., 2012] Giboney, J. S., Brown, S. A., and Nunamaker Jr., J. F. (2012). User acceptance of knowledge-based system recommendations: Explanations, arguments, and fit. *45th Annual Hawaii International Conference on System Sciences, Hawaii, January*.

[Gregor and Benbasat, 1999] Gregor, S. and Benbasat, I. (1999). Explanations from intelligent systems: Theoretical foundations and implications for practice. *MIS Quarterly*, 23:497–530.

[Harbers, 2011] Harbers, M. (2011). *Explaining Agent Behavior in Virtual Training*. PhD thesis, Utrecht University.

[Hempel and Oppenheim, 1948] Hempel, C. and Oppenheim, P. (1948). Studies in the logic of explanation. *Philosophy of Science*, 15(2):135–175.

[Jaws, 2012] Jaws (2012). Java api for wordnet searching (jaws). *Java API for WordNet Searching (JAWS) (http://lyle.smu.edu/ tspell/jaws)*, accessed 18-September-2012.

[Johnson, 1994] Johnson, L. (1994). Agents that learn to explain themselves. *Proceedings of the Telfth National Conference on Artificial Intelligence*, pages 1257–1263.

[Kayande et al., 2009] Kayande, U., de Bruyn, A., Lilien, G., Rangaswamy, A., and van Bruggen, G. (2009). How incorporating feedback mechanisms in a dss affects dss evaluations. *Information Systems Research*, 20:527–546.

[Mao and Benbasat, 2001] Mao, J. Y. and Benbasat, I. (2001). The use of explanations in knowledge-based systems: Cognitive perspectives and a process-tracing analysis. *Journal of Management Information Systems*, 17(2000):153–179.

[Miller, 1986] Miller, G. A. (1986). Dictionaries in the mind. *Language and Cognitive Processes*, 1(1986):171–185.

[Moulin et al., 2002] Moulin, B., Irandoust, H., Blanger, M., and Desbordes, G. (2002). Explanation and argumentation capabilities: towards the creation of more persuasive agents. *Artificial Intelligence Review*, 17(3):169–222.

[Mpagouli and Hatzilygeroudis, 2007] Mpagouli, A. and Hatzilygeroudis, I. (2007). Converting first order logic into natural language: A first level approach. *In Theodore S. Papatheodorou, Dimitris N. Christodoulakis, and Nikitas N. Karanikolas, editors, Current Trends in Informatics: 11th Panhellenic Conference on Informatics, PCI 2007*, A:517–526.

[Pieters, 2011] Pieters, W. (2011). Explanation and trust: what to tell the user in security and ai? *Ethics and Information Technology*, 13(1):53–64.

[Prakken and Vreeswijk, 2002] Prakken, H. and Vreeswijk, G. (2002). Logics for defeasible argumentation. *Handbook of Philosophical Logic, second edition*, 4:219–318.

[Reiter and Dale, 2000] Reiter, E. and Dale, R. (2000). *Building Natural Language Generation Systems*. Cambridge, UK: Cambridge University Press.

[Roth-Berghofer and Cassens, 2005] Roth-Berghofer, T. and Cassens, J. (2005). Mapping goals and kinds of explanations to the knowledge containers of case-based reasoning systems. *Berlin: Springer*, 3620:451–464.

[Salmon, 1989] Salmon, W. (1989). Four decades of scientific explanation. *Scientific Explanation*.

[Shieber et al., 1990] Shieber, S., van Noord, G., Pereira, F., and Moore, R. (1990). Semantic-head-driven generation. *Computational Linguistics*, 16(1):30–42.

[Shortliffe and Buchanan, 1975] Shortliffe, E. H. and Buchanan, B. G. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23(3–4):351–379.

[SIGGEN, 2012] SIGGEN (2012). Acl special interest group on natural language generation. *ACL SIGGEN - What is NLG? (http://www.siggen.org/nlg.html)*, accessed 24-August-2012.

[Simon, 1996] Simon, H. A. (1996). *The Sciences of the Artificial*. MIT Press, Cambridge, MA, third edition.

[Swartout and Moore, 1993] Swartout, W. and Moore, J. (1993). *Second-Generation Expert Systems*, chapter Explanation in Second-Generation Expert Systems.

[Swartout, 1983] Swartout, W. R. (1983). Xplain: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21.

[Swartout et al., 1991] Swartout, W. R., Paris, C., and Moore, J. (1991). Explanations in knowledge systems: design for explainable expert systems. *IEEE Intelligent Systems*, 6(3):58–64.

[Swartout and Smoliar, 1987] Swartout, W. R. and Smoliar, S. W. (1987). On making expert systems more like experts. *Expert Systems*, 4(3):196–207.

[Swinney, 1995] Swinney, L. (1995). The explanation facility and the explanation effect. *Expert Systems With Application*, 9(4):557–567.

[Taylor et al., 2006] Taylor, G., Knudsen, K., and Scott Holt, L. (2006). Explaining agent behavior. *Behavior Representation in Modeling and Simulation (BRIMS)*, 2006.

[Toulmin, 1958] Toulmin, S. (1958). The use of argument. *Cambridge University Press, Cambridge*, 1958.

[van Deemter et al., 2003] van Deemter, K., Theune, M., and Krahmer, E. (2003). Real vs. template-based natural language generation: a false opposition? *Computational Linguistics*, 31(1):15–24.

[van Lent et al., 2004] van Lent, M., Fisher, W., and Mancuso, M. (2004). An explainable artificial intelligence system for small-unit tactical behavior. *American Association for Artificial Intelligence*, (2004).

[Wallis and Shortliffe, 1982] Wallis, J. and Shortliffe, E. (1982). Explanatory power for medical expert systems: studies in the representation of causal relationships for clinical consultations. *Methods of Information in Medicine*, 21:127–136.

[Wang and Benbasat, 2007] Wang, W. and Benbasat, I. (2007). Recommendation agents for electronic commerce: Effects of explanation facalities on trusting beliefs. *Journal of Management Information Systems*, 23:217–246.

[Wedekind, 1988] Wedekind, J. (1988). Generation as structure driven derivation. *Proceedings, 13th International Conference on Computational Linguistics (COLING-88)*, pages 732–737.

[Weizenbaum, 1966] Weizenbaum (1966). Eliza - a computer program for the study of natural language communication between man and machine. *Computational Linguistics*, 9(1).

[Wick and Thompson, 1992] Wick, M. and Thompson, W. (1992). Reconstructive expert system explanation. *Artificial Intelligence*, 54(1-2):33–70.

[WordNet, 2012] WordNet (2012). Wordnet - a lexical database for english. *About WordNet (http://wordnet.princeton.edu)*, accessed 6-September-2012.

[Ye, 1990] Ye, L. R. (1990). User requirements for explanation in expert systems. Unpublished Doctoral Dissertation, University of Minnesota.

Thanks

There are some people that have directly or indirectly contributed to the creation of this thesis that deserve attention. First I wish to thank Alan Turing Institute - Gerard, Henk-Jan, Koen, Gerda, Daniel, Bas, Ivana, Bo, Jeroen, Anneke - for welcoming trainees so warmly and providing sufficient resources for this research. In particular, I would like to thank Sjaak for his jokes and distractions when we shared a room in the office. Thanks also to John Jules, my university's supervisor, for his support in tips and feedback and accepting me as his trainee in the first place. I thank Chide for his guidance during the research. I enjoyed the many appointments when we discussed the bigger and ethical questions; I found these discussions very valuable. I want to thank my friends and family who indirectly contributed greatly to this work. Finally, I wish to thank my girlfriend for her everlasting patience and belief in me and my work, I could not possibly have done it without you.

Nomenclature

AI	Artificial Intelligence
FOPL	First Order Predicate Logic
HeMAS	Heterogenous Multi Agent System
JAWS	Java API for WordNet Searching
KBS	Knowledge-Based Systems
NL	Natural Language
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
SFOPLR	Simple First Order Predicate Logic Reasoner
WFF	well-formed formula

List of Figures

1.1 Example MYCIN Rule in Lisp 13
1.2 Translation MYCIN Rule in English 13
1.3 XPLAIN Justifying 14

2.1 The Toulmin Model 20
2.2 The Coherence of the Six Toulmin Elements 21
2.3 Natural Language Processing: Two directions 22
2.4 Architecture of a NLG system 24

3.1 Example HeMAS Configuration 31
3.2 Example Fictional Abstract Explanation 31

4.1 Solution Overview 33
4.2 Example Bayesian Network 38
4.3 Example Restricted Toulmin Model for Bayesian Transition . . . 38
4.4 Toulmin Chain 39
4.5 Class Diagram for Toulmin Model 40
4.6 Example Logical Formula Decomposed 43
4.7 Translating FOPL to NL 44
4.8 Example Usage of XML for Templates 48
4.9 Example Explanation for FOPLR 50

5.1 LogicBabelfish Illustration 54
5.2 LogicBabelfish Handle Query 58
5.3 LogicBabelfish Handle Definition 59
5.4 LogicBabelfish Handle Interpretation 60

Index

alan turing institute almere, 10
anonymous call, 61
argument mapping, 20
argumentation theory, 19
artificial intelligence, 10
atomic agent, 27, 34

bayesian network, 34, 38
bdi-model, 14, 15
boltzmann machine, 34

call, 61
callpath, 62
canned text, 23, 36
clara, 30, 33, 36, 40
click-through hyperlinking mechanism, 29
click-through mechanism, 32, 39
click-through-mechanism, 34
cloudy agent, 34, 35
computational linguistics, 22
config agent, 27, 33, 35, 75
cyclic dependency, 62, 63

deep thought, 9
definition, 54, 55, 58–63, 65, 69
definition call, 61, 65
definitions, 56
dependency, 62

explain agent, 30
explainable agents, 30
explainable ai, 14, 17, 29
explanation effect, 18
explicit cyclic dependency, 63

first order logic, 41
first order predicate logic, 34

formpart, 55, 56, 63

grammar, 23

hemas, 10, 15, 18, 27–30, 33–36, 73
hempel-oppenheim-schema, 19
heterogenous multi agent system, 10

implicit cyclic dependency, 63
interlingua for explanations, 40
interpretation, 56, 66
interpretationpart, 56, 65

java api for wordnet searching, 46
justification, 19

knowledge-based systems, 17

lexicon, 46
limpid agent, 34, 35
logical form, 56, 59–62, 65
logicbabelfish, 34, 48, 53–58, 61, 63, 65, 71
logicbabelfish application, 55
logicbabelfish base, 34

marama, 11
micro planner, 23
micro planning, 24, 54, 68
mycin, 13–15, 17

natural language, 22
natural language generation, 22
natural language processing, 22
natural language understanding, 22
nlu, 71
non-anonymous calls, 61

priority, 65

prolog, 41

recursion, 63
restricted toulmin model, 37, 39

sfoplr, 49
simple first order predicate logic reasoner, 49
strategy explanation, 19
surface realisation, 54, 68
surface realiser, 24

terminological explanation, 19, 31
text templates, 23, 36
toulmin base, 34
toulmin chain, 39
toulmin model, 20, 34, 36, 39, 40
toulmin structure, 20
trace, 19
transition, 35

well-formed formula, 43
wordnet, 46

xml, 48
xplain, 14, 15, 17, 29