DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
**72**

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
**72**

**MARGUS TREUMUTH**

# A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects

TARTU UNIVERSITY PRESS

Institute of Computer Science, Faculty of Computer Science, University of Tartu, Estonia.

Dissertation is accepted for the commencement of the degree of Doctor of Philosophy (PhD) on May 19, 2011 by the Council of the Institute of Computer Science, University of Tartu.

Supervisors:

Prof. PhD Mare Koit
University of Tartu
Tartu, Estonia

Prof. PhD Kristiina Jokinen
University of Helsinki
Helsinki, Finland
Visiting Prof. of the University of Tartu

Opponents:

Prof. PhD Ramón López-Cózar Delgado
University of Granada
Granada, Spain

Senior researcher, PhD Hele-Mai Haav
Tallinn University of Technology
Tallinn, Estonia

The public defense will take place on June 29, 2011 at 16:00 in Liivi 2–404.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.



ISSN 1024–4212
ISBN 978–9949–19–701–9 (trükis)
ISBN 978–9949–19–702–6 (PDF)

Tartu Ülikooli Kirjastus
www.tyk.ee
Tellimus nr 346

# Contents

# List of Original Publications

1. Huang, H.-H.; Cerekovic, A.; Tarasenko, K.; Levacic, V.; Zoric, G.; Treumuth, M.; Pandzic, I.S.; Nakano, Y.; Nishida, T.[1] (2006). An agent based multicultural user interface in a customer service application. *In: Proceedings of the eNTERFACE'06 Workshop on Multimodal Interfaces: eNTERFACE'06 The SIMILAR NoE Summer Workshop on Multimodal Interfaces; Dubrovnik, Croatia*, 12–21.
2. Treumuth, Margus; Alumäe, Tanel; Meister, Einar[2] (2006). A natural language interface to a theater information database. *In: Language Technologies, IS-LTC 2006: Proceedings of 5th Slovenian and 1st International Conference, Ljubljana, Slovenia*, 27–30.
3. Treumuth, M. (2006). A Natural Language Interface to a Theater Information Database. *In: SPECOM'2006 Proceedings: XI International Conference Speech and Computer, St. Peterburg,* 179–181.
4. Treumuth, M. (2006). Dialoogsüsteemid – kuupäevade tuvastamine ja vastusemallid. Keel ja arvuti. Tartu Ülikooli Kirjastus, 210–220.
5. Treumuth, M. (2007). A Method for Recognizing Temporal Expressions in Estonian Natural Language Dialogue Systems. *In: Proceedings of the 16th Nordic Conference of Computational Linguistics: NODALIDA 2007, Tartu, Estonia,* 265–268.
6. Treumuth, M. (2008). Automatic Extraction of Time Expressions and Representation of Temporal Constraints. *In: Proceedings of the Third Baltic Conference on Human Language Technologies, HLT 2007: The Third Baltic Conference on Human Language Technologies; Kaunas, Lithuania;* 311–317.
7. Treumuth, M. (2008). Normalization of Temporal Information in Estonian. *In: Text, Speech and Dialogue, 11th International Conference, TSD 2008, Proceedings: Text, Speech and Dialogue, 11th International Conference, TSD 2008; Brno, Czech Republic;* Springer, 2008, (Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence; 5246), 211–218.
8. Treumuth, M. (2010). A Framework for Asynchronous Dialogue Systems. *In: Frontiers in Artificial Intelligence and Applications: HUMAN LANGUAGE TECHNOLOGIES — THE BALTIC PERSPECTIVE; Riga, Latvia;* IOS Press, 107–114.

[1] Contribution in paper No 1: shared ideas and experiments, contribution 10%.
[2] Contribution in paper No 2: shared ideas and text writing, equal contribution

# CHAPTER 1

# Introduction

## 1.1 The Overview

A dialogue system is a type of user interface (UI) where linguistic phenomena such as verbs, phrases and clauses act as UI controls for selecting data in software applications. In addition, the dialogue system is intended to converse with a human, with a coherent structure. Such an interaction involves the basic properties of human conversation, including turn-taking, initiative, significant silence and even manners.

The most frequent use of dialogue systems entails automatically answering questions posed in human language. To find the answer to a question, a computer program may use a pre-structured database or a web page (whether local or remote).

The thesis is about text-based human-computer conversations on the internet, where the user input is a written request to the dialogue system in a natural language and the output of the system is an answer to the user in the same language.

The goal of the work is to develop concepts to analyze dialogue systems on a uniform base, so that these concepts can be used to design and implement a framework for building dialogue systems – modular software that can be easily adapted to different domains.

The author of the thesis has implemented the Asynchronous Dialogue System framework (ADS framework) – a software system that consists of a collection of integrated modules, including several Natural Language Processing (NLP) modules that can be used in developing text-based natural language dialogue systems.

The ADS framework is currently tailored for Estonian language, yet most of its features and modules are easily transferable to English language. The dialogue systems based on the ADS framework mimic the natural interaction between people better than the models used so far.

## 1.2 Related Work

The available software for building dialogue systems is usually limited to a special domain or to a special language (mostly English) or to a special modality (e.g. spoken language). There are several projects that are similar to the ADS framework. Yet, none of them offers web-based asynchronous turn management and *human-assisted chat*[*]. No evidence of such implementations was available. The following sections provide reviews about the other dialogue system frameworks.

### 1.2.1 CSLU Toolkit

The Center of Spoken Language Understanding Toolkit (CSLU Toolkit) [CSLU Toolkit, 2009] [Sutton et al., 1998] was created to provide the basic framework and tools to build, investigate and use interactive language systems. The CSLU Toolkit incorporates speech recognition, natural language understanding, speech synthesis and facial animation technologies.

However, these modules of CSLU Toolkit are not language independent. Also, the CSLU Toolkit is not easily portable to the web.

The CSLU Toolkit is an example of a finite-state dialogue manager. It does not scale well for more complex applications or interactions. For instance, in a mixed-initiative system (where the user is also allowed to direct and shift the focus of the conversation), the number of transitions in the finite-state automaton grows very large; the representation becomes difficult to handle.

### 1.2.2 AIML

Artificial Intelligence Markup Language (AIML) [AIML, 2009] is an XML dialect for creating natural language software agents. The XML based AIML was developed by Richard Wallace and a worldwide free software community between the years of 1995 and 2002. Free AIML sets in several languages have been developed and made available by the user community. There are AIML interpreters available in Java, Ruby, Python, C++, C#, Pascal, and other languages.

AIML is highly portable, however, the *pattern matching*[*] in AIML is too limited. It permits only one wild-card (*) match character per pattern. AIML has no regular-expression support. Maybe the goal was ultimate simplicity at the price of functionality. Pattern matching with one wild-card symbol is not flexible enough. The frameworks that are based on AIML also do not have asynchronous turn management.

[*] see Appendix E for definition.

### 1.2.3 VoiceXML

VoiceXML [VoiceXML, 2009] [Lucas, 2000] is another example of a finite-state dialogue manager. VoiceXML (VXML) is the W3C's standard XML format for specifying interactive voice dialogues between a human and a computer. It allows voice applications to be developed and deployed in an analogous way to HTML for visual applications. Just as HTML documents are interpreted by a visual web browser, VoiceXML documents are interpreted by a voice browser.

As in CSLU Toolkit, the flow of the interaction in VoiceXML is described via a finite-state automaton. At each point in the dialogue, the system is in a certain state (each state typically corresponds to a system prompt). In each state, the system expects a number of possible responses from the user; based on the received response, the system transitions to a new state. To develop a dialogue management component for a new application, the system author must construct the corresponding finite state automaton.

VoiceXML is well suited only for implementing relatively simple systems that retain the initiative throughout the conversation. In these cases, the finite-state automaton representation is very easy to develop, interpret, and maintain. VoiceXML is not supported by common web browsers like Internet Explorer, Firefox and Opera.

### 1.2.4 Olympus/RavenClaw

Olympus [Bohus et al., 2007] is a dialogue system framework; RavenClaw [Bohus and Rudnicky, 2003] is the dialogue manager that acts in this framework.

Olympus/RavenClaw is a freely available framework for developing dialogue systems. It has been deployed by several dialogue systems in various domains including

- ConQuest – the system utilized at the Interspeech conference,
- RoomLine – the system is used to reserve rooms on campus,
- BusLine – a system for Pittsburgh bus information.

Olympus is based on the Galaxy Communicator. Each dialogue module (speech recognition, text-to-speech, parser, back-end, dialogue manager, etc.) runs as a separate server and communicates through the Galaxy hub. In the default setup, Olympus uses Sphinx2, Festival, the Phoenix parser, the Rosetta natural language generator, and the RavenClaw dialogue manager. Any module of the system can be replaced. In theory, any application can be wrapped to be an Olympus server. Servers (and wrappers) can be developed in C, Java, Python, Lisp or Perl.

RavenClaw, a dialogue manager server, adds value for a dialogue designer. RavenClaw is developed in C++. There is a logical separation between the system independent dialogue manager implementation and the design of a

particular dialogue system. The dialogue system designer needs no skills in C or C++ programming. The system specific part of the dialogue manager is defined using preprocessor macros. The dialogue is specified as a tree, with pre and post-conditions, and execution effects defined for each node. The leaf nodes of a dialogue tree are the agents responsible for some finite task of a dialogue: getting information from a user, presenting information, or accessing a database. During the runtime, the tree nodes are placed on the stack and executed in the order determined by each agent's pre-conditions. The dialogue designer specifies the tree nodes where each tree system allows grammar rules to be enabled in particular agents of the dialogue tree.

Yet, Olympus/RavenClaw did not seem suitable for these goals:

- to explore the advantages of asynchronous turn taking;
- to have a tight integration with a relational database;
- to have an availability for *human-assisted chat*[*].

### 1.2.5 Semantra

Semantra is a commercial Natural Language Interface (NLI) framework [Semantra, 2009] [Elder, 2004] for building search tools that let non-technical users make ad hoc queries in plain English.

The principle parts of the solution include:

- a semantic engine that parses natural language;
- a collection of ontologies and business rules that provide context;
- a dynamic query generator that creates the appropriate SQL (structured query language) command to be executed against a targeted database.

These foundational elements form a "semantic layer" between knowledge workers and enterprise applications with their relevant corporate data source(s). Semantra's natural language processor is the core semantic engine responsible for parsing users' common-language requests. The user submits the inquiry by typing into a search box. The NLP breaks down the sentence structure, interprets the grammar and phrases, handles the synonyms and part-of-speech elements, and even resolves misspellings on the fly. Should any semantic mismatches or *ambiguities*[*] still exist, the NLP assists the user in clarifying the request.

Once parsing is complete, the inquiry is mapped against the OntoloNet™, a hierarchical repository of business concepts, terminology and business rules that collectively form the backbone of Semantra's technology. This expanding and adaptable *knowledge base*[*] allows most any business to quickly bootstrap semantics into their enterprise while inheriting any shared (non-proprietary) concepts, business rules, jargon, and acronyms from Semantra's vertical industry

[*] see Appendix E for definition.

ontologies. These concepts and terms, along with target database metadata, are captured during Semantra's pre-deployment process, known as "semantification".

Once the inquiry is fully understood, the final step is to dynamically convert the request into the appropriate query command (SQL or related format) which is executed against the structured enterprise database(s). Within seconds of the initial request, the results of the query are displayed in the user's browser, in the selected graphical or tabular format. Additionally, query results can be exported to third-party applications such as business intelligence systems, report beautifiers, charting or mapping tools, or spreadsheets.

As Semantra is a commercial closed-sourced framework, it is not available for general development activities such as development of dialogue systems.

### 1.2.6 Other Frameworks

[Popescu et al., 2003] have presented a domain independent framework that can map natural language questions to SQL queries and have implemented a system PRECISE based on that framework. They prove that, for a broad class of semantically tractable natural language questions, PRECISE is guaranteed to map each question to the corresponding SQL query. They report on experiments testing PRECISE on several hundred questions drawn from user studies over three benchmark databases. They find that over 80% of the questions are semantically tractable questions, which PRECISE answers correctly. PRECISE automatically recognizes the 20% of questions that it cannot handle, and requests a paraphrase. However, there was now description about the amount of work involved in porting the NLIs between domains using PRECISE. Their system seems to do the porting automatically which is not convincing. The demo is not available. There is no way to tell whether PRECISE is a framework for building conversational agents or an attempt to handle question answering problems.

[Cimiano et al., 2007] have also presented a new model for user-centered adaption of NLIs to a certain domain. The model assumes that domain experts without any background knowledge about computational linguistics will perform the customization of the NLI to a specific domain. In fact, it merely requires familiarity with the underlying knowledge base as well as with a few basic sub-categorization types.

They have implemented a system called ORAKEL, which is a natural language interface to ontologies and knowledge bases. It was designed to be portable to different domains and knowledge bases. It provides a tool called FrameMapper which can be used to graphically map natural language expressions to relations defined in the knowledge base, thereby customizing the system for a specific knowledge base. Currently, ORAKEL supports two Semantic Web formalisms: F-Logic and OWL/SPARQL.

This section presented an overview of the available/accessible frameworks for building dialogue systems. The analysis of these frameworks led the author of this thesis to formulate the problem. In addition, the following chapters contain references to various related theoretical and/or technical solutions that have been used in building the ADS framework.

## 1.3 The Problem

The preliminary questions that triggered the work done in this research were:

1. Is there a suitable framework available for Estonian language that could be used in building dialogue systems? The aim is to have a generic framework with reusable components that could be adjusted for several domains with little programming effort.
2. Do any of the related frameworks have the support for asynchronous communication pattern and *human-assisted chat*[*]?

After the study of the related work, the problem arose, as none of these frameworks were sufficient to be used due to the following reasons:

- lack of pre-processing (including *stemming*[*] or morphological analysis) for Estonian language, and no support for custom pre-processing adjustments;
- no support for human-assisted chat;
- no support for web based asynchronous communication pattern.

Consequently, the task of this research was formulated – to implement a new framework that would be suitable for Estonian language, including:

- asynchronous communication,
- optional human assisted chat interface,
- support for Estonian *morphology*[*],
- support for Estonian *temporal expressions*[*],
- tight integration with a relational database,
- automated correction of spelling errors.

The need for pre-processing in Estonian language is due to the rich morphology. None of these tools offered an easy option to integrate a morphological analyzer into the system. Therefore, it would be rather uncomfortable to build systems for Estonian (or any other agglutinative language) with these tools.

The need for human-assisted chat and asynchronous communication pattern is based on exploitation issues. In simple restricted domains, the dialogue system can usually satisfy the information needs of the user. Yet, even in the sim-

[*] see Appendix E for definition.

ple domains it is unrealistic to expect that the capabilities of the system can handle all user requests correctly. It mostly leads to user disappointment when the system fails to perform as expected.

The ADS framework provides a hybrid approach – "a human assisted dialogue system" that allows a single human agent to handle a number of simultaneous chat sessions by having an AI-engine (the module that aims to implement the Artificial Intelligence (AI) abilities, incl. natural language processing) handle the bulk of common, repeat questions. The AI-engine will allow the human agent to focus his or her attention on the few chat sessions needing unique service and will effectively lower the cost of supporting chat sessions. The server-side technology of the ADS framework uses an AI-engine as well as a live agent backend interface to deliver live-agent experience without the user having to know whether the answer is from the AI-engine or from the human agent. The asynchronous communication pattern is an essential element in this approach, as both parties (human and computer) can provide input at any given moment.

This approach allows us to put these dialogue systems into practical use and avoid user disappointment. It can be compared to machine assisted translation, where part of the translation is done by the translation program and part by a human. The reason is the same – the translation programs are not good enough to fully satisfy the needs of the users.

Although, the dialogue systems developed in the ADS Framework, can be assisted by a human, still the goal of this research has always been maximizing the AI-participation in the conversation and minimizing the human intervention.

The human assisted approach is not a goal itself – it is merely providing the system developer with a constant flow of real data, as the dialogue systems can be brought out of the lab environment into real usage. The incoming real data from the real users leads to the development of better natural language understanding algorithms and conversational management. Also any small improvements to the system can be easily made available for real users.

Two dialogue systems have been built based on ADS framework. One of them (a virtual dental consultant Zelda) has been evaluated in practice by real users. The other dialogue system (movie schedule information provider Alfred) is a prototype and has been tested mostly by students or random visitors.

## 1.4 Major Contributions

The contribution of the author of this thesis is the implementation of the ADS framework, including all the components and methods listed below that were not available in other DS frameworks:

a) Asynchronous turn-taking strategy, so that both parties (human and computer) can provide input at any given moment and can take any number of

sequential turns without waiting for the other party to acknowledge each turn.

b) AI-assisted live agent chat, so that the unanswered questions can be handled by an optional human operator.
c) A language independent solution for the word-order problem, thus allowing skipping the syntactic analysis and optionally ignoring the word-order problem in the *knowledge engineering*[*] process. This is essential for languages with relatively free word order (such as Estonian).
d) A collection of temporal constraints for Estonian temporal expression recognition.

In addition, the author created the following features of the ADS framework that can be found in other DS frameworks:

a) A web-based conversation interface with optional speech synthesis.
b) Separation of declarative domain knowledge and procedural code. The domain specific knowledge and temporal constraints are separated from the central dialogue management.
c) Robust language analysis, so that the misspellings in the user input are corrected by the system. This method also includes the stemming (the process of reducing a word to its root word), to ease the pattern creation in knowledge engineering.
d) Easy and compact representation of knowledge, so that the domain adaptation and knowledge base engineering would contain a minimal amount of programming effort. The knowledge is represented as a set of pattern-response pairs. The system also includes pattern-function pairs to represent procedural knowledge.

The novelty of this work is in implementing a complex framework, containing all the components and features listed above, and exploiting this framework in building dialogue systems for Estonian language, thus effectively reducing programming effort in this task. The thesis is not so much about the techniques used, but about combined application and deployment of these techniques for a higher level task.

In the implementation of the ADS framework the author has also used the software that was created by others:

a) morphological analyzer of Estonian (by language software company Filosoft),
b) speech synthesis of Estonian (by Tallinn University of Technology and The Institute of the Estonian Language),
c) common software solutions (such as PHP [Atkinson and Suraski 2003], AJAX [Eichorn, 2006], Oracle [Loney, 2004], etc), which are all referenced in the thesis.

[*] see Appendix E for definition.

## 1.5 Thesis Outline

The rest of the thesis is organized as follows:

- Chapter 2 outlines the essential features of the ADS framework, including the *semantic resolution*[*] of user input and turn management.
- Chapter 3 describes the Estonian language and NLP modules in the ADS framework, including spell-checking, word-order and normalization of temporal expressions.
- Chapter 4 describes the domain adaptation issues and knowledge base creation.
- Chapter 5 discusses the implementation details about the system design: the user interfaces, the server modules and the database.
- Chapter 6 outlines the application issues and provides evaluation of the ADS framework.
- Chapter 7 discusses future research and Chapter 8 concludes the thesis.
- Appendix A defines a list of acronyms and gives an overview of technical specifications.
- Appendix B contains tables of noun cases and verb inflections of Estonian language.
- Appendix C lists some pattern-response pairs from the knowledge bases of ADS based dialogue systems.
- Appendix D contains sample conversations with the ADS based systems.
- Appendix E provides definitions to some basic terms that were used in the thesis.

## 1.6 Acknowledgements

I would like to thank my supervisors – professor Mare Koit and professor Kristiina Jokinen. I am very grateful to Mare Koit for her guidance, patience and understanding during my studies, and for providing consistent encouragement and assistance. I am also very grateful to Kristiina Jokinen who provided valuable suggestions and discussions. I would like to thank my wife Ene-Renate and my children Karmen and Sander for their constant support. I thank my parents, Peeter and Anneli, for having faith in me.

This dissertation was financially supported by the Center of Excellence in Computer Science (EXCS), the Tiger University Program of the Estonian Information Technology Foundation, the Estonian Science Foundation projects 7503 and 5685, by the National Programme for Estonian Language Technology projects EKT5, EKKTT09-57, EKKTT06-15.

[*] see Appendix E for definition.

# CHAPTER 2

# The ADS Dialogue Management Framework

This chapter outlines the essential features of the ADS framework. It starts with a brief overview of author's previous systems. Then an overview of the dialogue management engine of the ADS framework is given. The section continues with descriptions of the relational model of language analysis, the semantic resolution, word order issues, handling of repetitions, turn-management concepts, and dialogue task specification.

## 2.1 Introduction

The previous dialogue systems of the author [Treumuth et al., 2006] were implemented for the Estonian language without much emphasis on reusable components. These systems were author's first attempts to implement a dialogue system for restricted domain with open prompt approach. The only similarities of these previous systems with the current ADS framework are:

- the usage of morphological analyzer in the pre-processing step,
- the usage of the *base forms*[*] in the semantic resolution,
- the usage of speech synthesis.

One of the previous systems Teatriagent [Treumuth et al., 2006] was also integrated with the speech recognition component [Alumäe, 2006] (the speech recognition was tested only in lab conditions, not over the Internet in public use).

In the ADS framework the author has dropped the speech recognition component as its availability to the general public is still limited. Instead the author has implemented a new turn management approach which is suitable only for text-based dialogue systems. This asynchronous communication pattern is described in Section 2.6.

[*] see Appendix E for definition.

In addition, some other noteworthy features that were not used in the previous systems and that became available with the ADS framework, are discussed later in this thesis, including:

- a solution to the word order problem in semantic resolution,
- a solution for spell checking the user input,
- a solution for the live agent assistance.

These features are discussed later in the thesis.

## 2.2 The ADS Framework Architecture

The complete client-server model of the ADS framework is described in Chapter 5. In this section the overview of the dialogue management engine is given. See Figure 2.1 for the diagram of the dialogue management engine.

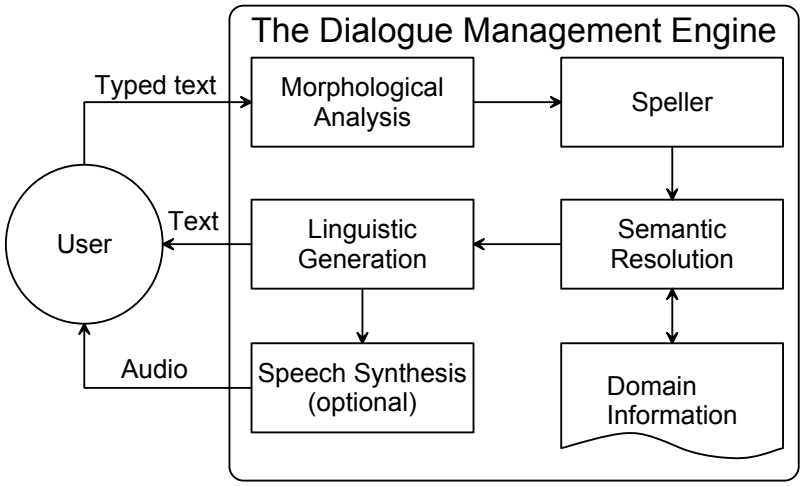



**Figure 2.1:** The Dialogue Management Engine of the ADS framework.

The morphology module is used for extracting the base forms from the user utterance (described in Section 3.2). This module integrates the morphological analyzer for Estonian language – Estmorf [Kaalep and Vaino, 2001].

The speller module is used for the correction of typing errors. This is a language independent module implemented by the author and is described in more detail in Section 3.3.

The semantic resolution is described in this chapter. The linguistic generation is used mainly with temporal expression and a brief description of this is given also in this chapter. Domain information is the rule-based knowledge base which is referred to through the thesis.

Speech synthesis is an optional external component which is discussed in Section 5.3.1. The technical details of speech synthesis can be also found in [Treumuth et al., 2006] and [Meister et al., 2003].

## 2.3 The Relational Model in Language Analysis

Resolving the user input involves parsing the input stream and placing its contents into a relational model.

The relational model is shown below in Figure 2.2. All the attributes and constraints have been removed from the diagram to keep the diagram readable. Only the entities and relations have been kept.

All relations are "one-to-many". A one-to-many relationship means that one row in one of the tables will relate to many rows in the other table. The Crow's Foot notation is used in relations – identifying the many, or child, side of the relationship, using the crow's foot at the line endpoint.

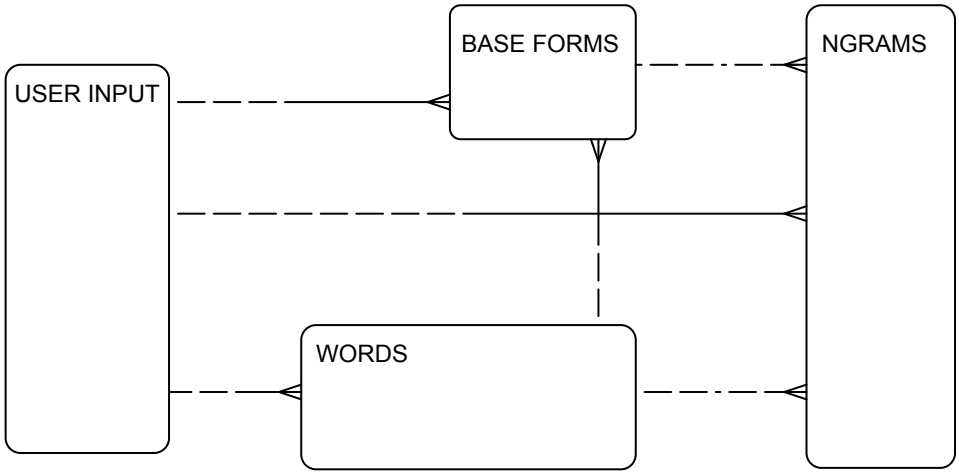



**Figure 2.2:** The entity-relationship diagram for the pre-processing stage.

This entity-relationship diagram implements the following relations:

- A user utterance can contain more than one word, base form or *n-gram*[*].
- N-grams are generated from the user input and are initiated by words or base forms.
- One word or a base form can initiate more than one n-gram.
- Words can have more than one base form.

[*] see Appendix E for definition.

Entities (User Utterance, Words, Base forms, N-grams) are implemented as tables. Relations are implemented by referential integrity constraints (primary keys and foreign keys). This part of the entity-relationship model is session based. This means that many simultaneous conversations (sessions) can be held with the dialogue system. The session support is implemented by session reference attributes in all entities.

There is a considerable amount of pre-processing done to reach this model (tokenization, morphological analysis and spell checking). After the system has completed this model by filling out the tables, the semantic resolution of user input can begin.

## 2.4 The Overview of the Rule Based Semantic Resolution

There are two main rule-based approaches in the ADS framework for semantic resolution. The first approach resolves the semantics of basic key phrases. The second approach resolves the semantics of temporal expressions. Both of these rule-based approaches use a declarative representation and the knowledge base consists of pattern-response pairs.

This separation of knowledge bases is rather similar to the approach used by [Dzikovska et al., 2003] where they decided to separate the knowledge used in general purpose language parsing from the knowledge used in reasoning. This section describes the similarities and differences of both approaches and provides an essential overview in understanding the process of the user input resolution as a whole.

### 2.4.1 The Resolution of Basic Keyphrases

The structure of the rules is given as:

```
RULE
   PATTERN – a regular expression*
   RESPONSE – a static response
   STATE – reference to additional responses
   IGNORE_WORD_ORDER – ignore word order (Y/N)
```

The reference to additional responses (attribute `STATE`) can be blank. The patterns are given as regular expressions. The pattern may contain just one keyword. The switch for ignoring word order of the input phrase (`IGNORE_WORD_ORDER`) is explained in more detail in Section 3.4.

* see Appendix E for definition.

The sentences for answering are given as predefined fixed sentences. The ADS framework also uses dynamic responses that are generated based on the information retrieved from the database. Yet, these dynamic responses are not represented in the declarative knowledge base. They are represented as procedures.

An example of a rule with attribute `STATE` undefined:

```
RULE
    PATTERN: (kartma|hirm) (valu|arst)
    RESPONSE: Ei ole põhjust karta!
    STATE: <undefined>
    IGNORE_WORD_ORDER: Y

RULE (translated)
    PATTERN: (scared|fear) (pain|doctor)
    RESPONSE: There is no reason to be scared!
    STATE: <undefined>
    IGNORE_WORD_ORDER: Y
```

There are numerous pre-processing steps and morphologic transitions that are applied to the user input prior to the semantic resolution. The semantic resolution uses the knowledge base to find the suitable answer.

The search for a suitable answer starts by matching the patterns of the rules to the pre-processed user input. The relational model from Figure 2.2 is expanded by a global entity RULES in Figure 2.3 to show how the rules are attached to the user input after the semantic resolution has been completed.

The expanded entity-relationship diagram (ERD) in Figure 2.3 shows that a pattern of a rule can match:

- a word,
- a base form,
- an n-gram.

After matching the patterns of the rules to the pre-processed user input, the according response sentence (or set of sentences) is selected. The selected sentence or set of sentences is forwarded to the planning module. The planning module decides whether and how to use this sentence or set of sentences in replying to the user.

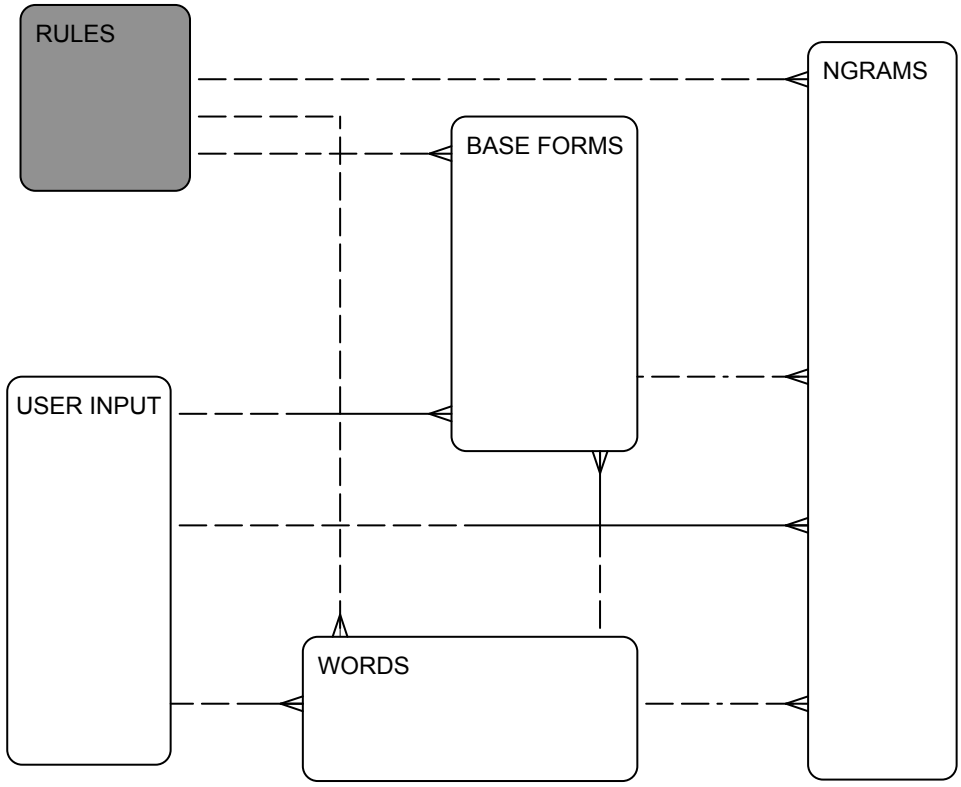



**Figure 2.3:** The entity-relationship diagram for resolving the user input.

### 2.4.2 The Resolution of Temporal Expressions

The general structure of the rules is given as:

```
RULE
   PATTERN – a regular expression
   CONSTRAINT – an SQL constraint
```

The structure is similar to the structure of the rules of basic key phrases. The patterns of temporal expressions are also given as regular expressions, exactly as in the basic key phrase approach. Yet, the constraint is not given as a fixed constraint. The constraint contains back references that are dependent on the regular expression of the temporal expression. In addition, the constraint is formulated as an SQL expression. This also means that the constraint cannot be used directly as a response.

For example:

```
RULE
  PATTERN – kell (\d{1,2}\:\d{2})
  CONSTRAINT – to_char(kuupaev, 'hh24:mi') =
lpad('\1', 5, '0')

RULE (translated)
  PATTERN – at (\d{1,2}\:\d{2}) o'clock
  CONSTRAINT – to_char(date, 'hh24:mi') = lpad('\1', 5, '0')
```

In this example, there is a back reference `'\1'` which value is dependent on the regular expression sub-pattern `(\d{1,2}\:\d{2})`. This back reference is evaluated and a final SQL constraint is formed in order to pass the query to the database.

After the pre-processing steps and morphological transitions the similar search is performed over these rules. If a match for a temporal expression is found then the according SQL constraint is selected.

However, the selected SQL constraint is not forwarded to the planning module. Instead, it is forwarded to a certain semantic resolution module for identifying compound temporal expressions. The SQL constraint may be joined with other existing SQL constraints to form a compound constraint.

The final SQL constraints are forwarded to the query generation module. The query generation module concatenates the constraints with the query templates.

The queries created by the query generation are executed on the domain specific timetable (movie schedule, train schedule, etc), which is structured as a calendar:

```
CALENDAR
   TIME – date
   EVENT – a static name of an event
   PLACE – a location of the event
```

For example:

```
CALENDAR
   TIME: 14.05.2010 17:00
   EVENT: Avatar
   PLACE: CINAMON
```

The queries find all `EVENTS` which meet the `TIME` criteria specified by the constraint. (The event “Avatar” is the name of a movie and “Cinamon” is the name of a movie theatre.)

### 2.4.3 Triggering Functions upon Matched Patterns

Most of the knowledge is expressed as pattern-response pairs. Most of the system responses are predefined sentences (facts that are listed in the knowledge base). The exceptional cases of *knowledge representation*[*] involve search for temporal information from the schedule.

However, much of the knowledge cannot be expressed as pattern-response pairs. The ADS framework is improved to use the pattern-function pairs in the knowledge base, in addition to the pattern-response pairs. This provides an option to the ADS framework to execute a procedure (function) based on any matched pattern. This option can be used in cases where finding an appropriate response involves some procedural knowledge. Then an appropriate procedure could be triggered to determine an appropriate response.

The ADS framework includes an additional column in the RULES table that holds the name of the procedure to be triggered upon a matching pattern.

### 2.4.4 The Conclusions of the Rule Based Approach

The experiments with the ADS framework have shown that within a restricted domain the framework has proven to work well. As previously said, two dialogue systems have been built with this framework. One of them has been in public use since 2008.

These experiments confirm that the rule based semantic analysis that uses pattern-response pairs in the knowledge representation is a reasonable and effective approach.

The key phrases describe the knowledge of the domain. The process of gathering domain specific knowledge and creating the rules for the knowledge base involves administrative work. The representation of patterns by regular expressions can require special skills. This process is described in more detail in Section 4.1.2.

Yet, the process of understanding the user input is not merely handled by the rule based semantic analysis. In addition, the pragmatic analysis is involved in the conversation. For example, the system understands and reacts appropriately, when:

- the input from the user is a repeated input,
- there has been too long (2 minutes) pause between two inputs,
- there has been long enough (5 minutes) pause between two repeated inputs.

The features of this pragmatic analysis establish the core competencies of the framework and are described in the following section.

[*] see Appendix E for definition.

### 2.4.5 The Core Competencies of the ADS Framework

There are certain conversational skills in the ADS framework that are built in the system core. The title "Core Competencies" applies rather well to these features. Also "Situation Policies" or "Timing Policies" are relevant titles for such procedural knowledge. [Bohus et al., 2007] have referred to such features also as "universal dialogue mechanisms". These features are domain-independent and partially language-independent and cannot be expressed by declarative rules in the knowledge base. The representation of this knowledge is procedural. These are also the features that most Eliza-like dialogue systems are lacking.

These features mainly depend on:

- time;
- the frequency of the user input;
- the user input patterns;
- previous dialogue (i.e. dialogue history).

Examples of these features can be:

- knowing the date and time;
- knowing what has already been said in a conversation (e.g. the system is able to repeat and is able to avoid repetitions);
- ability to tell when the user repeats itself and ability to respond to the user repetitions;
- knowing how to react to a long pause;
- ability to understand that after a certain amount of mishandled inputs, the system has failed to reply, ability to provide an appropriate reaction in case of such failure (e.g. by admitting its failure and offering an apology);
- ability to understand if the user is very active or rather inactive, ability to react to such situation;
- ability to understand that the user is just testing the system by entering only single keywords and not using full sentences, ability to react to such situation;
- ability to understand if the user's sentences are too long, asking the user to rephrase in shorter words;
- ability to understand that the conversation has been going on for a long time, ability to react to such situation;
- ability to understand if asked to be silent, to speak more quickly, to talk more slowly;
- ability to adjust the turn-taking pace by the user's writing speed
- knowing how to start a conversation from the beginning (e.g. by flushing conversation memory upon a specific command from the user).

These features have a specific role in the conversation. They describe the awareness of the situation in which the dialogue system is. They also describe how to respond to these situations. The aim is to keep the conversation smooth,

to assist and to appear intelligent. They can be considered as a sub-task on "Problem Solving" stage.

Looking at the classic model of a dialogue system, there are three main stages:

1. Language Analysis
2. Solution to the problem
3. Language Generation

The sub-features of a dialogue system built on the ADS framework can be grouped below these stages as follows:

Language Analysis:

- morphological analysis, finding the base forms of words;
- spell checking;
- normalization of calendar expressions;
- language identification.

Problem Solving:

- **core competencies** (or conversational skills)**;**
- query generation;
- querying the database.

Language Synthesis:

- generating sentences;
- speech synthesis.

As the text-based conversations with the ADS framework take place on the internet, the language used can be a bit different at times. For example, the users of the dialogue system usually do not use capital letters. They also tend to use short phrases similar to spoken language. The dialogue systems built on the ADS framework have also been adjusted to look natural for the internet chat. For example, the system responses also drop the capital letter from the beginning of the sentence.

## 2.5 Handling Repetitions in Conversation

This section describes the repetition problem. In any conversation, both the system and also the user can repeat a previous utterance. The ADS framework does not set a limit to the user repetitions. However, the ADS framework sets a limit to the system repetitions.

In the spoken language conversations the repetition can be a part of the repair strategy. For example, the user might not have heard what was said in the conversation and therefore specifically requests for a repetition (possibly by

also repeating itself). In this case, it is appropriate for the system to repeat the previous utterance.

However, in the text based conversations, the user can always scroll back in the chat history and look at the whole conversation. So, in the text based conversations the repetition by the system is usually not needed and should be avoided or used only after a period of expiration to prevent user frustration.

In order to avoid unnecessary repetitions by the system, there should be a clear understanding why these repetitions occur. There are two main reasons that are causing the repetitions by the system.

Firstly, the system repetitions can be triggered by the user who has repeated a previous utterance.

A simple example would be:

| | |
|---|---|
| System: | Good morning! |
| User: | Hello. |
| System: | Hello. |
| System: | I can provide information about ... |
| User: | Hello. |
| System: | Hello. |

Secondly, the repetitions can also occur if the system produces the same output from the different inputs, which is a less likely scenario. For example, various greetings from the user could be replied with a certain single phrase, thus repeating the same greeting.

An example would be:

| | |
|---|---|
| System: | Good morning! |
| User: | Hi. |
| System: | Hello. |
| System: | I can provide information about ... |
| User: | Hello. |
| System: | Hello. |

The repetitions in both cases are usually considered to be a sign of poor intelligence of the system. [Vrajitoru, 2006] has said that the repetition decreases the life-like impression of the system and undermines the credibility of the system.

The repetition testing is also the most frequent test done by the first-time users. Many first-time users are quite often testing the system capabilities by saying the same greeting more than once. If they see that the system returns the same greeting the exact same amount of times, their respect towards the system is decreased. Yet, if the system does not fall for this test, the attitude of the user is better in the following conversation as the system has passed a basic test of intelligence.

A simple way to avoid repetitions by the system would be to check the chat history prior to using a phrase in replying to the user.

In case the planned reply is found in the chat history, one of the following actions could be taken:

a) the reply will not be issued and the system will be silent;
b) the reply will not be issued and the system will use another reply if there are any other pending replies in the stack;
c) the same reply will be rephrased and issued;
d) the same reply will be issued referring to the recurrence of this reply (e.g. "as I previously said, …").

The ADS framework currently uses options a) and b).

In addition, the expiration interval is used by the ADS framework. This means that the repetitions are only those recurring replies that are not older than two minutes from the previous occurrence. If the same reply is older than two minutes, it is not considered to be a repetition of the reply and it is issued as a regular reply. This way the repetition is less disturbing as some time has passed since the previous output.

Also, the ADS framework does not simply search the chat history for the previous occurrences of a phrase, because in the ADS framework the response might be a combination of many phrases and the chat history contains a full version of the response. This is explained by the following example.

Let's have rules $R_1$ and $R_2$, so that

$R_1$ would produce output $W_1$;

$R_2$ would produce output $W_2$.

If both of these rules

$$R_1 \text{ and } R_2$$

are matched simultaneously, then the system could very likely produce a concatenated output $S_1$ by adding the phrases together (usually separated by a space character and a comma):

$$S_1 = W_1 \cup \ldots \cup W_2 .$$

The concatenated answer $S_1$ would be stored in the chat history. This means, that $W_1$ and $W_2$ are not stored in the chat history as two distinct replies.

If the rule $R_2$ is matched later again in the same conversation, then the reply $W_2$ would have to be issued. Yet, it has already been issued as part of $S_1$. The problem is that $W_2$ is not found in the chat history on its own. This is why the simple search in the chat history cannot be used in avoiding repetitions.

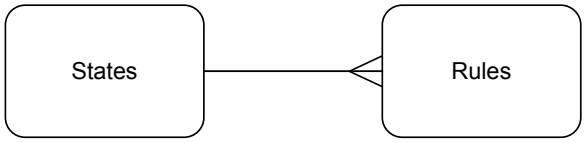
We could search for a substring from the chat history and see that $W_2$ is a substring of $S_1$, as

$$W_2 \subset S_1 .$$

Yet, the search for a substring would also not be a valid approach in many cases. For example, the reply "OK" could be a substring of many previous responses. So, $W_2$ could even be a substring of $W_1$.

Therefore, instead of searching the chat history, a separate log (rule history) is used, which records the rules that have been resolved and used during the conversation. This way it is easy to see, that the rule $R_2$ has been used in the conversation less than two minutes ago and therefore the reply $W_2$ will not be issued.

Similar logs are kept about the states and templates. We could have skipped the log of states, and only used the log of rules. Yet, the rule and the state are not in one-to-one relationship. One state could be triggered by many rules, as in:




The template history is also a completely separate history. The templates are used to provide variations while generating the response by rephrasing a certain part of the sentence. The same phrasing is avoided by keeping the history of previous phrasings.

## 2.6 Turn Management

An essential feature of a dialogue system is turn taking. The spoken language systems and text-based systems both handle turn taking usually in synchronous communication pattern. These dialogue systems assume a rigid (you speak – I speak) turn-taking behavior.

**The main problem with this approach is that the system has to wait for the user to provide input and cannot give additional information.** User usually does not have to wait for the system, as in such dialogue systems the reply to the user is usually immediate (depending only on the time that it took to process the input and find the answer).

In the ADS framework the synchronous communication pattern is replaced by asynchronous communication pattern. We never consider a user input ending point as passing the turn to the computer. The user can keep on giving input at any time. This input is all stored and the computer can answer at any time.

As in a normal text-based chat – all parties can speak at any given moment and can take any number of sequential turns without waiting for the other party to acknowledge each turn.

The asynchronous communication pattern also has some advantages in Wizard-of-Oz (WOZ) data collection [Rieser and Lemon, 2008] and in live-agent assisted chat. The pause in the asynchronous communication pattern is not a sign (give-away) that the computer has been replaced by a human, because the computer has been making pauses all along the way.

The user of a dialogue system with synchronous communication pattern can be tricked into WOZ assistance too, but long pause is not normal in dialogue systems with synchronous communication pattern. The long pause in text-based synchronous dialogue systems is rather rare. The user has to be explained why it is sometimes taking so long to reply while usually the answer came in just a few milliseconds. So it is complicated to switch to WOZ or live-agent assistance in dialogue systems that are using the synchronous communication pattern.

The new turn management approach – asynchronous communication pattern – which is suitable for text-based dialogue systems was implemented in the ADS framework and is described in this section.

### 2.6.1 The Serial Synchronous Communication Pattern

The importance of turn management is often underestimated in dialogue systems. The emphasis in modeling dialogue systems is mostly on providing an intelligent answer to the user. Most of these systems however are totally helpless and stay quiet without the user input. Most dialogue system architectures are either pipelined or are restricted to a pipelined flow-of-information.

The following is a conversation structure with the synchronous turn-taking and it follows the pattern:

Human=>Computer=>pause=>
Human=>Computer=>pause=>
…
Human=>Computer

Normal human-to-human text-based chat that lasts more than a minute hardly ever takes this form. This is unnatural and disregards the central theme and advantages of natural language interfaces – their natural feel and look. The unnatural model "forgets" that the main advantage of intelligent user interfaces compared to direct manipulation systems (point-and-click interfaces) is the natural and easy communication. This advantage is lost when the user has to communicate in an unnatural style and has to be constantly active participant in the conversation.

In the synchronous turn-taking, only one of two collaborators (either system or user) issues a unit of text and then waits for the input of the other. This turn-taking is quite unnatural in written internet conversation. Yet, this is mostly the case with text-based dialogue systems. These systems are able to provide rather intelligent answer at times, yet the conversational pattern is very limited.

### 2.6.2 The Asynchronous Communication Pattern

The ADS framework provides the asynchronous communication pattern. Then the turn-taking process refers mostly to the sequential submission of elaborated units of information, namely collaborative contributions. In this case, quite complex patterns of turn-taking may evolve:

Human=>Computer=>Computer=>short pause=>
Computer=>Human=>short pause=>
Human=>Human=>longer pause=>
Human=>short pause=>
Computer=>Computer

The ADS framework allows real-time user-initiated interruptions, which gives an impression of a natural conversation.

The analogy to normal human behavior is not the only benefit of this communication pattern. Asynchronous turn-taking opens up many ways to enhance the dialogue, e.g. the DS can:

- answer many questions at once,
- provide system initiative recommendations [Misu et al., 2010],
- acknowledge a question while it is still working on finding the answer [Blaylock et al., 2002],
- inform the user of a new, important event, regardless of whether it is tied to the user's last input [Allen et al, 2001].

The technical implementation of the asynchronous communication is achieved by using AJAX (see Figure 2.4). The browser (client) is sending one initial request and then starts two background processes at certain intervals: one for sending data and the other one for receiving information. The server is also constantly monitoring the process (whether there is input or pause) and creates responses accordingly.

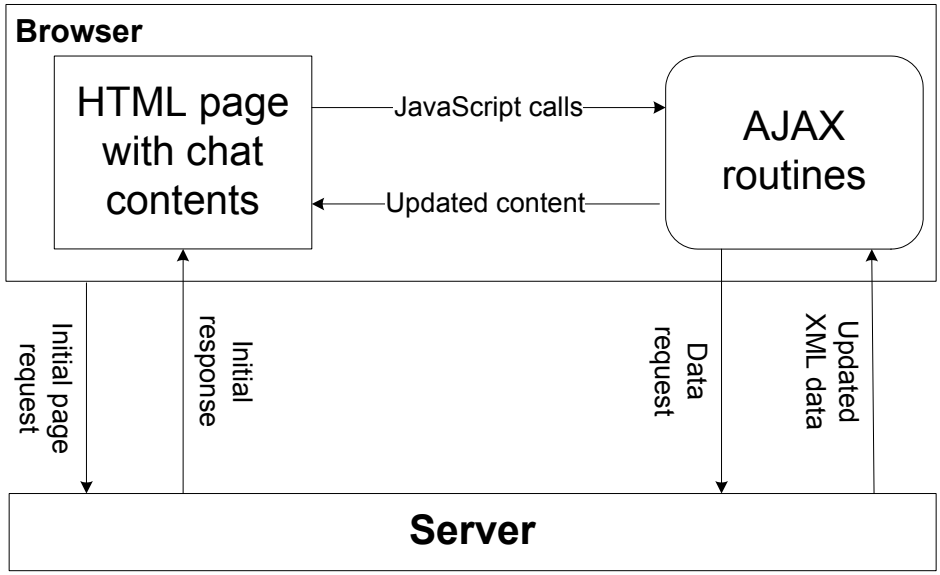



**Figure 2.4:** The implementation of asynchronous communication in the ADS framework.

### 2.6.3 The Asynchronous Communication and Wizard-of-Oz

One of the benefits of the asynchronous communication pattern is revealed in the process of WOZ data collection.

It is known that the user is quickly adjusting to the partner in the conversation [Stenchikova and Stent, 2007]. This also means that there are certain differences in the human-to-human conversations and in the human-to-computer interactions.

We want to model the human-to-human communication, so we need to collect the data that is similar to the human-to-computer interaction.

Therefore, while performing the WOZ data collection, it is essential to hide the fact that the computer is replaced by a human. If the user discovers the trick, the dialogue might change into a complex human-to-human communication. This we do not want to happen. We need to keep the secret from the user while collecting sample conversations. Yet, this might be impossible while using dialogue systems that are running in serial synchronous communication pattern.

If the dialogue system is running in the serial synchronous communication pattern, then the main problem in the WOZ data collection, is the high predictability of the turn-taking pace. The user can easily guess when and how quickly the system usually would reply, because the serial synchronous systems always use a fixed turn-taking pace.

For example, after a few turns the user knows that the system usually replies within 1 second. Most of the dialogue systems that are using serial synchronous

communication pattern, usually reply in less than three seconds. Often the reply is given immediately without any delay. The only delay is generated due to slow computing environment (the time spent on parsing the request) and in some cases a short delay also appears due to the slow network traffic. These systems never change this turn-taking pace during the whole conversation.

If the wizard (a human) takes over the conversation, it is very hard to maintain this high fixed turn-taking pace. The human needs more time to find the answer and more time in typing the response. The time spent on replying is not quick and not with fixed rate any more – it is slower and irregular during each turn.

In the WOZ data collection also some additional phrases (such as "Please wait!"; "Hold on, please") are used by the wizard to extend the time for information retrieval. These additional phrases with irregular and longer pauses can unfortunately be a quick give-away. If the system used to have a certain turn-taking rate before the wizard took over, then additional pauses and phrases, such as – "Please wait!" – might appear suspicious to the user. If the user suspects that the partner in the conversation is not a computer but a human, then the user usually starts a more complex conversation and the data collection results reflect the human-to-human conversations.

However, in the asynchronous communication pattern we have much better chances to trick the user into believing that the partner is still the computer. As stated before, in the asynchronous communication pattern:

- the user can enter input at any given moment;
- the system can reply at any given moment.

In this case, the additional and irregular pauses are typical and accepted by the user. There has not been any fixed turn-taking rate that the user could have memorized. So, the user has no grounds for any suspicions based solely on turn-taking pace.

The other matters that could lead to suspicions (like change in style and complexity) are irrelevant at this point in discussion as they can appear in both kinds of systems, independently from the turn-taking issues.

## 2.7 The Dialogue Task Specification

The dialogue task specification describes an overall plan for the interaction. The ADS framework provides a way to make some minor domain-specific adjustments to the plan of interaction, yet the fundamental execution pattern is fixed and not adjustable without additional programming effort.

# CHAPTER 3

# Natural Language Processing Modules

The ADS framework consists of several independent NLP modules. At first, this chapter gives a brief overview of Estonian morphology and syntax as the ADS framework is currently tailored for Estonian language. After that the following NLP modules are described:

- morphology module [Kaalep and Vaino, 2001],
- spell-checking module,
- normalization of temporal expressions.

## 3.1 Handling Estonian in Language Analysis

The ADS framework accepts the user input without any constraints and limitations. The system does not restrict the user and does not present prompts with a selection of limited answers. This approach is also known as the open prompt approach or non-restrictive approach [Jurafsky and Martin, 2000]. The restrictive approach would be to constrain the user to some specific response, such as: "Say *yes* if you accept the booking, otherwise, say *no*".

The grammar in the non-restrictive approach must recognize any kind of response, since the user could say anything. This brings us to the problem of linguistic complexity and this is when the morphology and syntax become an important issue.

Estonian is an agglutinative language, which means that the morphemes that contain grammatical information are appended to the word stems (mostly as suffixes). The stem of the word can also be modified in this process. Therefore the Estonian language is also an inflected language. For example, the Estonian illative case is expressed by a modified root: *vesi* 'water' → *vette* 'into the water'.

In Estonian compound words can be formed to express complex concepts as single words. For example, the words *abi* 'help' and *palve* 'request' can be combined to form a word *abipalve* 'help request'. In Estonian, more than two stems can be added together, which is rather rare in English. Estonian nouns have 14 cases while English has only two cases. The 14 noun cases and verb inflections are listed in Appendix B.

In Estonian language:

- neither nouns nor pronouns have grammatical gender;
- there are no words that consist of only one letter [Alumäe, 2006].

The word order in Estonian is relatively free. Sometimes the words in a sentence can be reordered without a change in the meaning of the sentence. The word order problem is handled by the ADS Framework as described in Section 3.4.

## 3.2 The Morphology Module

The morphology module of the ADS framework integrates the morphological analyzer of Estonian [Kaalep and Vaino, 2001]. The morphological analyzer is used in the preprocessing step to generate base forms from the original word forms. After this step the user input is stored in two different versions:

- an original version
- a version with base forms

The version with original word forms has the priority over the version with base forms in the pattern matching process. In case the original form is successfully matched to the knowledge base patterns, then the version with base forms is ignored.

It was a rather complex technical task to integrate this analyzer with the ADS framework. The morphological analyzer is a command line tool written in C. The binary is wrapped with Java [Arnold and Gosling, 1996] and packaged in Oracle as a Java package. In addition, the file system is used for storing temporary input and output files.

The resulting interface is a simple Oracle PL/SQL function which provides a seamless way to use the morphological analyzer straight in the Oracle database with PL/SQL. This multi-step integration looks like an unstable integration, yet it works without any problems.

## 3.3 Spell-Checking and Error Correction

The spell checking approach in the ADS framework is language independent and developed by the author of this thesis.

While studying the conversation logs of the dialogue systems that were built using the ADS framework, it turned out, that approximately 80% of the users make spelling errors in major keywords. As the keywords are essential in understanding the user input, the ADS framework implements a basic spell checking as a pre-processing step in resolving the meaning of the sentence. In one of the dialogue systems developed with the ADS framework (a dialogue

system called "Zelda") the number of spelling errors corrected was 380 for the total of 4500 user utterances (that is 8.4% of utterances were corrected).

The spell checking approach used in the ADS framework can be thought of as context-sensitive approach that is exploiting string similarity.

### 3.3.1 Jaro-Winkler vs Levenshtein

The most well known string similarity metric is Levenshtein distance [Black, 2005], yet it is not good enough for spell checking.

The Jaro-Winkler distance [Cohen et al., 2003] is used in the ADS framework while spell checking the user input. The Jaro-Winkler distance is a measure of similarity between two strings. This is implemented as a function:

$$Similarity = Jaro\_Winkler(string\_1, string\_2)$$

and the return value of the function is normalized as

$$Similarity \in [0..1]$$

The similarity score 0 equates to no similarity and 1 is an exact match.

Jaro-Winkler distance [Cohen et al., 2003] uses a prefix scale which gives more favorable ratings to strings that match from the beginning for a set prefix length. This is the main advantage if compared to the Levenshtein distance [Black, 2005].

It is a good assumption that the mistake is not usually in the beginning of the word. With Levenshtein, the beginning of the word would be treated equally with the end of the word and the results would not be as good as with Jaro-Winkler.

For example, if the user input would be

***n**aistes (in women)*

and the lexicon would contain a word

***p**aistes (swollen)*

then the unwanted replacement by the Levenshtein would be made

$$naistes \Rightarrow paistes \text{ (both words were grammatically correct)}.$$

The unwanted replacement by the Levenshtein would occur as the number of edits would be the minimum similarity threshold – just one edit from the exact match. This is an unwanted replacement and Jaro-Winkler approach would not make this replacement. The Jaro-Winkler approach does not consider these two words similar (based on static similarity threshold 91.2% which was used in the ADS framework).

Yet, if the userinput would be

*paiste**b** (swollen – misspelled)*

and the lexicon would contain a word

*paiste**s***

then the replacement b y the Jaro-Winkler would be made

$$paisteb \Rightarrow paistes.$$

In both cases just one of the symbols was different. Yet, the beginning of the word is usually a bit more important as it defines the stem. Therefore, the first example was not acceptable by Jaro-Winkler algorithm.

### 3.3.2 Domain Lexicon

The domain lexicon (made from the pattern-response pairs and contains words from the patterns) is refreshed automatically daily at 6 AM. An automated task in the ADS framework generates a domain lexicon, based on the words that appear in the recognition patterns defined as regular expressions. This automated generation of the lexicon is necessary to minimize the risk that the administrator of the system updated the knowledge base (changed/added/removed the rules) yet forgot to update the lexicon.

Words shorter than 6 letters are not spell-checked as the ambiguity risk would be too high. So, these shorter words will not be added to the lexicon.

The lexicon is made of the words that appear in the rules. The reason is that there is no need to spell-check the words that the system does not "understand". The language analysis capability is limited to the rules. So, the rules contain all the words that we need to capture and understand.

### 3.3.3 Accuracy Score

The score above 0.912 turned out to be good enough for acceptance in correction. So the similarity threshold was set to 0.912 in the ADS framework.

**IF** the Jaro-Winkler similarity measure of a user input word **A** and a lexicon word **B** is greater than or equal to **0.912**; and **A** is not equal to **B**

**THEN** make the replacement:

$$A \Rightarrow B$$

The sentence before the replacement was:

$$w_1 w_2 \ \mathbf{A} \ w_4$$

The sentence after the replacement is:

$$w_1 w_2 \ \mathbf{B} \ w_4$$

The number 0.912 was decided by experiments.

## 3.4 Word Order Issues in Language Analysis

In the language analysis process, usually the grammar is adjusted to look for the phrases relevant to the domain [Jokinen et al., 2002].

The ADS framework uses a similar approach. The grammar in the ADS framework can be a simple grammar that is meant to extract just the single keywords and their relevant word forms, with the help from a morphological parser. In such a simple case, the word order is not an issue.

Yet, in a more complex task, the single keyword approach is not enough to capture the meaning of the sentence. Then, in addition to matching the single words, the phrases have to be matched and then the word order of a phrase becomes an issue. Usually more than two different wordings have to be considered to recognize the relevant phrase.

One of the options is to handle the word order of the sentence manually by defining all possible word order permutations in the grammar. In this case, the number of permutations can be rather high and usually only the most probable permutations are listed in the grammar to keep the grammar readable. The manual approach is also supported by the ADS framework. The word order can be ignored by adding manual word permutations into the rules of the grammar.

The problems with this manual approach are:

- readability of the grammar is decreased,
- the developer can forget to add the permutations to the grammar,
- the developer can provide an insufficient amount of permutations.

The ADS framework also includes an optional automated approach, as the manual approach is not always efficient and elegant as shown in the following examples.

Let us suppose that the phrase

$$w_1 w_2 w_3$$

could be accepted in all possible permutations:

$$w_1 w_2 w_3$$
$$w_1 w_3 w_2$$
$$w_2 w_1 w_3$$
$$w_2 w_3 w_1$$
$$w_3 w_1 w_2$$
$$w_3 w_2 w_1$$

By *acceptable permutation* we mean that the permutation of the phrase captures the meaning of the sentence. So, we would have the maximum of six acceptable permutations of a three word phrase, as $P_3 = 6$.

As the ADS framework uses regular expressions to define the patterns of phrases, the rule would be a regular expression listing all the six possible wordings:

$$(w_1 w_2 w_3) \mid (w_1 w_3 w_2) \mid (w_2 w_1 w_3) \mid (w_2 w_3 w_1) \mid (w_3 w_1 w_2) \mid (w_3 w_2 w_1)$$

The pattern from the example above seems rather readable, yet it is not mostly true. The problem is that $w_i$ can be a regular sub-expression (e.g. set of synonyms), as in the following example $w_1$ and $w_2$ contain a set of synonyms expressed as a regular expression:

$w_1$ = (hea|ilus|tore),

$w_2$ = (uus|algav).

To further illustrate this problem, we should look at a rule that is defined by a phrase with three words $w_1 w_2 w_3$ so that the phrase would also accept synonyms.

For example a pattern $p_1$ for $w_1 w_2 w_3$ could be:

| | |
|---|---|
| Example of a pattern | (hea\|ilus\|tore) (uus\|algav) aasta<br>*(good\|beautiful\|happy) (new\|starting) year* |

This pattern $p_1 \Rightarrow$ "(hea|ilus|tore) (uus|algav) aasta" is sufficient to match the input:

| | |
|---|---|
| Example of an user input | "head uut aastat"<br>*"Happy New Year"* |

However, there can be more than one acceptable permutations of this phrase. For example, this pattern is not enough to match any input where the word order is a bit different, such as:

| | |
|---|---|
| Example of an user input | "uut aastat ilusat" * |
| | *"starting year beautiful"* |

So, there is a need to write an additional pattern to match this order of words "uut aastat ilusat". The word order of the new pattern would be defined as:

$$p_2 \Rightarrow w_2 w_3 w_1$$

and would be written as:

| | |
|---|---|
| Example of a pattern | (uus \| algav) aasta (hea \| ilus \| tore) |
| | *(new \| starting) year (good \| beautiful \| happy)* |

At this point we can choose whether to have these two separate rules $R_1$ and $R_2$ for patterns $p_1$ and $p_2$ in the system to match the same meaning (which is not an elegant solution) or whether to concatenate these two rules $R_1$ and $R_2$ into a single long pattern listing all alternative word orders, such as:

((hea | ilus | tore) (uus | algav) aasta) | ((uus | algav) aasta (hea | ilus | tore))

From the example above, we can see that even by allowing just two different word orders, the simple pattern quickly becomes unreadable. And we still have described only two different word order patterns. We would have to concatenate all permutations for all the possible word order variations. There would be a maximum of six permutations for a three word phrase as $P_3 = 24$, and four word phrases would need 24 permutations as $P_4 = 24$.

As a solution to the word order problem, the ADS framework has an additional attribute (*IGNORE_WORD_ORDER*) in the definition of a rule to allow automated word permutations. This allows us to keep the patterns simple and still be able to handle the word order.

The attribute can have two values:

```
IGNORE_WORD_ORDER = [YES; NO]
```

This attribute also leaves an option to turn off automated permutations for a certain pattern. Automated permutations have to be closed for some patterns where changing the word order would change the meaning.

* Comment to the previous example: In Estonian, the word order of the previous sample phrase is acceptable in some sentences. Yet, it might be a bad example for English as this particular word order might be rather unusual.

The attribute IGNORE_WORD_ORDER = YES for all the rules where an arbitrary sequence of words does not change the meaning of what the rule needs to capture.

And respectively, IGNORE_WORD_ORDER = NO for all the rules where only a fixed sequence of words is allowed to capture the meaning.

Automating the permutations greatly simplifies the process of grammar design and keeps the grammar more readable. Without this method we would have to explicitly indicate word order variations by listing all possible realizations. With this new approach, if the rule is marked to allow free word order, all the variations of word order can be accepted.

How does it work? All permutations of an n-gram are considered in the matching process. The pattern of the rule will not be permutated because it is a very complicated task to permutate the regular expression. It is much easier to permutate the words in the n-grams.

If we receive an input phrase from the user such as “ma nägin sind juba” (meaning “I saw you already”), then all the 24 permutations of the phrase would be:

ma nägin juba sind
ma juba nägin sind
ma juba sind nägin
ma sind nägin juba
ma sind juba nägin
...
sind ma juba nägin
...
sind ma nägin juba

All these 24 n-grams would be considered in the matching process, and this way all word order variations are accepted with just one rule

```
RULE
    PATTERN: mina nägema sina juba
    RESPONSE: Tore, siis me oleme kohtunud!
    STATE: <undefined>
    IGNORE_WORD_ORDER: Y

RULE (translated)
    PATTERN: I see you already
    RESPONSE: Great, then we have met before!
    STATE: <undefined>
    IGNORE_WORD_ORDER: Y
```

Notice that all the words in the pattern above are in the base form (in the Estonian example the word *“nägema”* vs. *“nägin”, in translation “see”* vs. *“saw”*). This provides a way to accept all possible forms of a given word. See Section 3.2 for detailed information about the morphological analyzer.

The ADS framework has been provided with a list of predefined permutation keys that define all possible patterns. These predefined keys will be used as indexes to the words in the phrase, thus producing all the necessary permutations without major computational complexity.

The ADS Framework also includes an option to define *stop words*[*]. The removal of stop words can further improve the phrase search. Stop words are words which are filtered out prior to semantic resolution. There is not one definite list of stop words. It can be different for each domain.

## 3.5 The Resolution of Temporal Expressions

The resolution of temporal expressions was discussed in Section 2.4.2. Some additional aspects of this process are described in this section.

The resolution of temporal expressions is not used in all domains, so the ADS framework contains an optional component for this task. This component was developed previously by the author as the temporal information can often be a significant part of meaning communicated in dialogues. The component and related work is described in more detail in [Treumuth, 2008].

The normalization of calendar expressions in the ADS framework is implemented as a separate domain independent module and is integrated into the semantic resolution module. The database of the normalization model contains rules that are used in the process of normalizing the date expressions. (See Appendix C for a sample listing of the rules.)

Table 3.1 lists a few examples of calendar expressions and their normalization results, to give a basic idea of the temporal normalization. The normalization result is usually dependent on a given date, in this example the starting point or reference date is *March 21, 2008*. This reference date is used to determine which calendar day is meant by the expression. If no special reference date is given as a starting point then usually the current date is assumed to be the reference date.

**Table 3.1:** Examples of calendar expressions and their normalization

| Estonian | English | Normalization |
|---|---|---|
| 3 aastat tagasi | 3 years ago | 21.03.2005 00:00 |
| 14. veebruaril 2004 a | February 14, 2004 | 14.02.2004 00:00 |
| kell 17:00 | at 5 PM | 21.03.2008 17:00 |
| jõulud | Christmas | DAY=[24.12, 25.12, 26.12] |
| esmaspäeviti kell 8 | on Mondays at 8 AM | (DAY=Monday) AND (TIME=8:00) |

[*] see Appendix E for definition.

The ADS framework has implemented a task specific approach where the normalization is formulated as a constraint to be applied on a calendar in the database.

Conjunctive and disjunctive expressions are also formed into an SQL constraint. This means that a larger temporal unit is a concatenation of smaller temporal units. This provides a way to capture duration, intervals and recurrences.

The ADS framework implements an empirical judgment that the word "ja" ("*and*") in Estonian natural language is mostly used to indicate that it is meant to be handled as disjunctive rather than conjunctive expression. There remains some ambiguity in this approach, yet in most cases this empirical assumption has proven to be a correct choice.

The composition of rules is also based on the time granularity measures. The granularities *hour*, *day* and *month* are used. The change in the granularity level indicates that a conjunctive expression has to be created.

A task specific constraint relaxation is also used by the ADS framework in resolving the temporal expression. For example, the user might mention a date to a dialogue system that would result in "*not found*" response. Then it would be appropriate to relax this date constraint, as in the following dialogue.

<User>: Are there any performances on Saturdays?
<System>: No, yet I found one on this Sunday . . .

This was an example of a constraint relaxation where the original date constraint was relaxed by adding one day. This way the users of the system can receive some alternative choices, instead of plain "*not found*" responses.

The relaxation parameters are not rule-based and are defined in the procedures of the temporal resolution. The programming effort is required in adjusting the relaxation parameters.

# CHAPTER 4

# Domain Adaptation

This section discusses the adaptivity of the ADS framework to a new domain/topic.

The ADS framework has been built so that the domain dependent dialogue management and the generic dialogue feature handling are separated. This significantly decreases the development effort when the ADS framework has to be adjusted for a new domain. The domain shift does not mean that the entire knowledge base has to be replaced. There are many common phrases (e. g. greetings and small talk) that can be kept for the new domain.

The following three points are addressed in the domain adaptation process:

- The adjustments in the knowledge base: What are the requirements for changing the knowledge base? How well can we represent the knowledge of a target domain as pattern-response pairs?
- The adjustments in the dialogue management: What are the components of the ADS framework that can be changed?
- The adjustments in the user interface: titles, names and pictures.

## 4.1 Adjusting the Knowledge Base

Most of the domain specific logic is kept in the knowledge base. The overview of the semantic resolution is given in Section 2.4. It states that the rule based semantic resolution is used in the ADS framework and most of the knowledge is represented as pattern-response pairs.

The process of changing the knowledge base involves two main considerations:

- How well can we represent the knowledge of a target domain as pattern-response pairs?
- How to define the rules in the knowledge base?

### 4.1.1 Domain Adaptivity to Pattern-Response Pairs

Many forms of knowledge representation have been developed over the years for conversational interfaces. Rules, frames, scripts and semantic network are the typical examples of knowledge representation scheme [Sajja and Akerkar, 2010]. The most common form has been to represent the knowledge as pattern-response pairs.

It seems that a rather high number of domains could be represented as pattern-response pairs, or at least could be represented in this form to some extent.

Supporting this consideration, is the fact that there are many examples in non-conversational information systems where the knowledge is represented in the similar way. For example, many internet sites provide information in a form known as "Frequently Asked Questions", or FAQs. The FAQs are listed questions and answers, all supposed to be frequently asked in some context, and pertaining to a particular topic. All these FAQs are rather similar to pattern-response pairs.

Certainly, there can be domains that cannot be represented simply by FAQ models. Examples of such domains involve task-specific reasoning and domains with many variables and a dynamically changing database.

The feasibility issue is also rather important in the decision process for a new domain. For example, the experiments show that it is not feasible to provide information about timetables in conversational form. This information should be presented in table form. However, this is a rather subjective statement as the conversational form can be the only acceptable form in some cases (e.g. phone conversations).

### 4.1.2 Defining the Rules in the Knowledge Base

As we have previously stated in Section 2.4 the structure of the basic rules is given as:

```
RULE
   PATTERN – a regular expression
   RESPONSE – a static response
   STATE – reference to additional responses
```

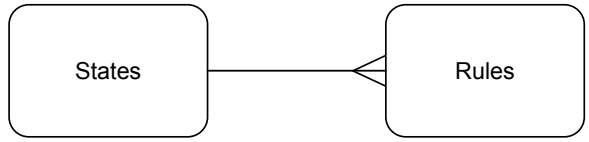
The pattern may contain just one keyword and the set of sentences can contain just one sentence. The patterns are given as regular expressions. The sentences for answering are given as predefined fixed sentences. The ADS framework also uses dynamic responses that are generated based on the information retrieved from the database. Yet, these dynamic responses are not represented in the declarative knowledge base. They are represented as procedures.

These pattern-response pairs are defined in the database tables *RULES* and *STATES*. The attributes of these tables are:

| Table "RULES" | Attributes |
|---|---|
| | pattern |
| | response |
| | ignore word order (y/n) |
| | reference to the state |

| Table "STATES" | Attributes |
|---|---|
| | name of the state (non-unique) |
| | response |
| | order of response |

The relation of these entities is "one-to-many", as in:




*"Triggering a state"* means that the response is made of many sentences that are presented to the user in an adjustable order.

As shown in the table description of *States* the name of the state is a non-unique attribute. This means that there can be many responses under the same name in the table *States*. This denormalization was more effective rather than implementing a 3NF many-to-many relationship.

The pattern first activates the rule and the rule issues the first response sentence and then triggers the state. This means that the information is provided in several parts. This leaves the impression for the user that the system remains in the topic with several successive turns providing the details of the same topic.

Most of the patterns are phrases where all the words are represented by their lexical base forms. This way fewer patterns need to be defined as in most cases the morphological form is not important for a pattern in the semantic resolution. The morphological analyzer adjusts the user input to the base forms as said above.

One pattern can contain a set of expressions in the same context. By the "same context" we mean that the phrases in the pattern are equal in the semantic sense and the same response can be used.

### 4.1.3 A Sample Process of Knowledge Engineering

A sample process of knowledge engineering is discussed in the following example. It makes no difference whether we would like to alter the knowledge base a new conversation topic or to expand the knowledge base for a new ques-

tion in the same domain. In both cases, there are two basic options to consider in the process of altering/expanding the knowledge base:

- to add a new rule;
- to alter an existing rule (for example, by adding new synonyms to the pattern).

In the following examples the discussion is mostly about the patterns in the rule and not about the responses of the rule.

If the ADS framework contains a rule with a pattern:

| | |
|---|---|
| Pattern | parkima sõiduk |
| | *park vehicle* |

Then the system is able to answer to the question:

| | |
|---|---|
| Question | Kuhu võin parkida sõiduki? |
| | *Where can I park the vehicle?* |

It is important to notice that the transitions:

*parkida → parkima*

*sõiduki → sõiduk*

are done with the morphological analyzer [Kaalep and Vaino, 2001].

Yet, the system is unable to answer the question:

| | |
|---|---|
| Question | Kuhu võin parkida **auto**? |
| | *Where can I park the **car**?* |

In this case, a new rule could be added to the system with the following pattern:

| | |
|---|---|
| Pattern | parkima auto |
| | *park car* |

Yet, there is also another option. The existing rule could be altered by adding the word “car” to the pattern. So, the existing rule would be changed to contain a pattern:

| | |
|---|---|
| Pattern | parkima sõiduk\|auto |
| | *park vehicle\|car* |

If we know that in this domain the users can talk about cars only when they are interested in parking information, then we could allow a less specific pattern:

| | |
|---|---|
| Pattern | sõiduk\|auto |
| | *vehicle*\|*car* |

Also the synonyms may be added as desired:

| | |
|---|---|
| Pattern | sõiduk\|auto\|masin |
| | *vehicle*\|*car*\|*machine* |

If, however, we know that in this domain, it is necessary to distinguish between the two topics regarding cars, then we need at least two rules with the following patterns:

For example:

| | |
|---|---|
| Pattern | parandama\|töökoda\|katki sõiduk\|auto |
| | *repair*\|*workshop*\|*damaged vehicle*\|*car* |

and

| | |
|---|---|
| Pattern | parkima\|jätma sõiduk\|auto |
| | *park*\|*leave vehicle*\|*car* |

The same way, if we know that there is no need to distinguish between repairing cars and repairing some other things, we could drop the cars from the pattern, as in:

| | |
|---|---|
| Pattern | parandama\|töökoda\|katki |
| | *repair*\|*workshop*\|*damaged* |

and

| | |
|---|---|
| Pattern | parkima\|jätma |
| | *park*\|*leave* |

This is not a very safe approach as we have the risk that the broad meaning of the words “damaged” and “leave” may be used in a different context that is not connected to the cars.

Therefore, it would be better not to remove the part of the rule referring to the cars.

## 4.2 Adjusting the Dialogue Management

The dialogue management engine has currently only one parameter that can be configured in domain adaptation:

- PARAMETER_INFORM_SUGGEST – an option to turn off the module INFORM_SUGGEST.

By turning off the INFORM_SUGGEST module, no suggestions are made to the user. For example, then the dialogue system is not using user pauses for giving system initiative recommendations, such as "*I would recommend to see the movie ...*".

## 4.3 Adjusting the User Interface

The graphical user interface (GUI) contains the name of the avatar (system character, such as our implementations Alfred or Zelda) and the picture of the avatar. It also contains a title of the system.

The GUI can easily be modified to meet the needs of a specific domain. Some basic knowledge of HTML [Raggett et al, 1999] and CSS [Lie and Bos, 1997] is required to make the necessary adjustments to the GUI.

## 4.4 Domain Adaptation Experiments

The ADS framework has been tested on the following three domains:

- conversation with a virtual politician,
- conversation with a virtual dental consultant,
- conversation with a virtual movie schedule administrator.

The first one (the virtual politician) was a simple test with some voluntary users. The second one (the virtual dental consultant) has also been tested in a commercial environment. The third example (virtual movie schedule information desk) is a natural language interface to a database. It serves as an interface to a dynamically changing database.

The first two domains are based on "Frequently Asked Questions" and static answers. The third domain is a more complex domain with a dynamic knowledge base.

The distinction between the complexities of the domains is fuzzy. It is hard to distinguish between the domains that are simple and those that are complex, in terms whether the ADS framework is able to accommodate to this domain. Yet, the domains with the static knowledge base can usually be considered to be less complex than those with a dynamic knowledge base.

# CHAPTER 5

# System Design

This chapter discusses the implementation details about the ADS framework: the user interface, the server modules and the database. Figure 5.1 represents the client-server model of the ADS framework.

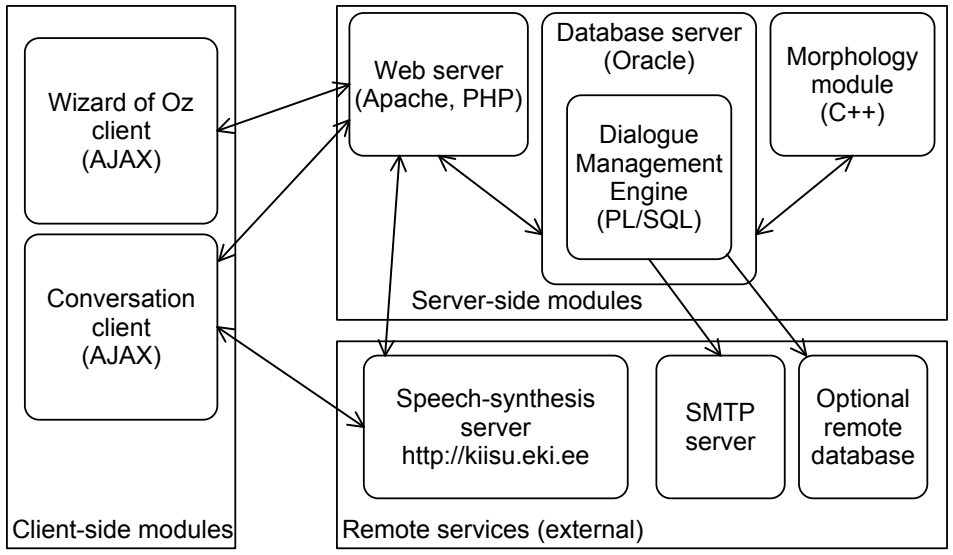



**Figure 5.1:** The client-server model of the ADS framework.

## 5.1 Client-side Code

The ADS framework is very much a server-centric model where client-side code is kept to a minimum. All updates and effects to the user are "pushed" to the client in response to an AJAX request. There are many reasons for this approach that are not relevant at this point, e.g. one of the reasons being code copy protection.

Even though the client is kept very thin, there are still many essential features of the user interface that are controlled by the client-side code. One of them is response timing.

There is currently no way to implement an HTTP [Fielding et al., 1997] push. This means that the server cannot initiate any transactions. The alternative would be using a thick client (applet/Flash [Allaire, 2002]) but the intention was to keep the client thin for better accessibility of the end-users (as applets or Flash might be disabled at the end-user's browser).

So, it is the client that initiates all transactions by polling the server for any new information. The client uses AJAX request to periodically poll the server. As it is essential to keep the incoming traffic to the server as minimal as possible, all AJAX requests between the client and server are reduced in frequency and size. The client polls the server at a variable rate – usually between 2 and 10 seconds (almost "real-time"). Also all redundant information is stripped from the package by the server and added later by the client. Multiple equal requests from the client are not prevented as it is very hard to anticipate what the server "wants" to reply to any request, as the server is keeping its own history of the conversation where timing and pauses can have a certain meaning and therefore can trigger certain events independently from the client. For example, the server might consider it appropriate to provide some additional suggestions if the client has been quiet for a long time.

### 5.1.1 Conversation Interface

The Conversation Interface is a thin AJAX client which communicates with the server via HTTP protocol [Fielding et al., 1997] over the Internet. Figure 5.2 represents the graphical user interface components of the conversation interface.

The GUI is built to be moderate and simple. There are no un-necessary control elements (no menus, no links) – just a text area with chat contents, an input area, and an option to turn ON/OFF voice (speech synthesis).

This GUI should be relatively intuitive for any user who has seen a common text-chat interface before.

It is easy to modify the GUI for any specific need. A developer with some basic knowledge of HTML and CSS can change any component of the GUI to meet the needs of a specific domain.

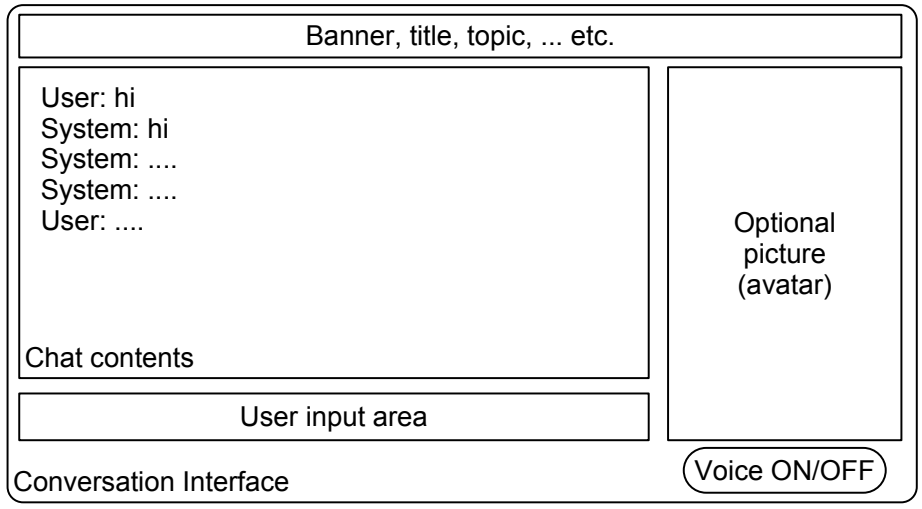



**Figure 5.2:** The GUI components of the conversation interface.

### 5.1.2 Wizard-of-Oz Interface

The WOZ interface is a thin AJAX client which communicates with the server via HTTP protocol over the Internet.

The WOZ interface supports unlimited number of wizards and unlimited number of conversations, while the number of conversations being more than or equal to the number of wizards.

The wizard will see a list of available conversations in the WOZ interface. The conversation becomes visible in the list if the user of the dialogue system has provided its first input in the dialogue. A conversation turns idle (become unavailable for the wizard), if the user has provided no input in 2 minutes. Idle conversation is not available for the wizard. Only available conversations are listed to all wizards. Figure 5.3 represents the graphical user interface components of the WOZ interface.

Each wizard can join any number of available conversations. Joining a conversation immediately eliminates access to the same conversation by the other wizards. This way the wizards do not get to join the same conversations. Only one wizard can be in any conversation. So, a conversation also becomes unavailable for other wizards after it has been joined by a wizard.

The process of joining is not reversible, that is – after joining, the wizard can not release a conversation to the other wizards. This would be a nice feature to have, yet is not implemented at this moment.

If a wizard has joined more than one conversation, a selection between these conversations has to be made before providing input to the conversation. The wizard can change this selection, so that all open conversations would receive some input from the wizard. This can be quite a difficult task to switch between

many ongoing conversations and keep track of all of them. It could be rather hard for a human assistant to handle more than five parallel conversations.

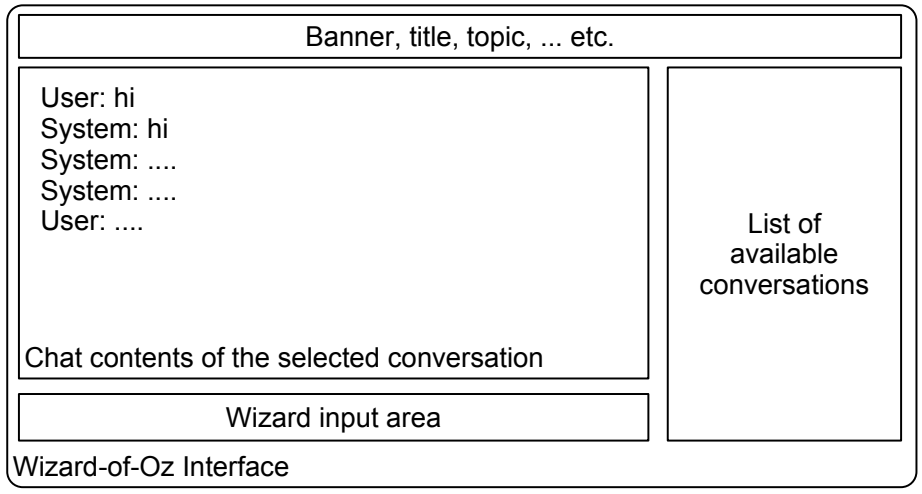



**Figure 5.3:** The GUI components of the Wizard-of-Oz interface.

While the wizard is helping the system, the system (i.e. the dialogue engine) is not turned into the silent mode. The system still keeps answering to the questions if able to do so. This can be quite helpful but can also lead into duplicate answers if the wizard is not aware of the system capabilities.

The interface would be better, if it could inform the wizard about the intentions of the dialogue engine (e.g. "The dialogue engine is willing to answer ..."). The planned answer of the system could be shown only to the wizard and the wizard can approve or disapprove the answer. This way the wizard could have control of the conversation and the system would not be able to interrupt. This feature of interruption control is not implemented yet and the system can interrupt at any moment. Usually this is not a problem, as the wizard is usually aware of system capabilities and is able to lead the conversation in a way that system can complete its responses and wizard can assist where necessary.

The WOZ interface is currently also able to inform the wizard about a starting conversation by sending a text message to the wizard's mobile phone. This way the wizard does not need to constantly monitor the interface for available conversations. This notification is sent by a SMTP protocol [Postel, 1982]. That means the system needs a SMTP server or has to have access to such a server. Currently the system uses an SMTP server that is freely provided by the internet service provider. This server can be replaced with any available SMTP server.

These notification messages are very useful, as the conversations with live systems can be rather rare and the wizard would have to spend too much idle time waiting for a conversation to start.

## 5.2 Server-side Code

Most of the code in the ADS framework is server-side code, the main reason still being the code security (copy prevention).

### 5.2.1 PHP Modules

Some PHP code is used to wrap the function calls to the database and to provide some basic pre-processing to the input by removing potential security risks from the input.

The PHP code in the ADS framework is also and essential element in session management. Sessions are initiated and handled by PHP. The session number is passed to the database upon each request along with additional input data from the client. Client-side session handling is covered by the client browser.

### 5.2.2 Database

Most of the key procedures in the ADS framework are implemented as stored procedures in the Oracle database. The choice of database (Oracle vs. MySQL [Widenius et al., 2002]) was a matter of experience. Any of these database engines could be used.

These key procedures include:

- input processing (including morphological analysis)
- finding the answer
- output processing

The main reason for implementing the key procedures in the database is the fact that the ADS framework stores most configuration parameters and data in the database. This data includes rules and templates. The database also contains all log files, including conversation logs and bug logs.

An alternative implementation would be to store data as text files on disk or in the database and keep the procedures as server-side code.

The main advantage of using stored procedures is the seamless integration with the data.

For example, there is no need to load the rules from the text file or database prior to the matching process. The rules are in the table and the processing is done by simply selecting the rules from the table.

There is also no need to make frequent connections to the text file or to the database to store conversation logs. The stored procedures can simply insert the sentences into the conversation log table. This keeps the number of connections to the database at minimal level.

The process of input parsing in the database is well observed as all the intermediate data during the parsing process is inserted into the database tables. The relational model is described in Figure 2.2.

The relational model provides a solid logical structure for the following relations:

- the words are associated with the base forms and n-grams
- the base forms are associated with the n-grams
- the rules are associated with n-grams, words and base forms

This data structure is created and stored for each conversation and is available during the entire session. These session references are also used during the WOZ process.

## 5.3 Remote Services

Currently two remote services are used by the ADS framework:

- Speech-synthesis server
- SMTP server

These services are non-essential in a sense that their unavailability or malfunction is not an issue for the system health as a whole. The system can continue without these services.

### 5.3.1 Speech-Synthesis Server

The speech-synthesis server is used to create speech output for the user. The speech output is a nice addition to the plain text output that is generated anyway.

The key point to mention in using the speech-synthesis option is that the server-side modules do the very minimal communication with the speech-synthesis server (by sending the server the sentence to be synthesized) and the most is done by the client-side code (by downloading the speech file to the clients machine real-time). This load-balancing is a crucial point as the ADS server would not be able to handle the requests for speech files and provide Dialogue Manager functionalities at the same time. This solution helps to keep the network traffic of the ADS server minimal.

The lack or malfunction of speech-synthesis would only affect the output in a way that the output would be plain text without speech. As the speech-synthesis option is left optional in the client module by a certain checkbox (speech ON/OFF). As the default setting is OFF, the user can be rather certain that this is not a crucial feature to have.

Currently the speech-synthesis server at http://kiisu.eki.ee/ is used. This speech-synthesis server has been created by Tallinn University of Technology

and the Institute of the Estonian Language [Meister et al., 2003], [Mihkla et al., 1999].

### 5.3.2 SMTP Server

The use of SMTP server is completely irrelevant to the end-user and is meant only for the wizards (the users of WOZ). The SMTP server is used to send notifications to the wizard's mobile phone to inform the wizard that a conversation has started.

The lack or malfunction of the SMTP server results in no notifications being sent to the wizard's mobile phone. This is rather bad as the wizards would then miss most of the conversations. No wizard usually sits and waits for a conversation to just happen. The frequency of conversations is usually rather low.

The system could be set up so that the invitation is sent only when the system is in trouble (that is – unable to find an appropriate answer). Currently, this option is not used and the notification is sent immediately after the user has issued a first sentence. There is also an expiration interval – that is notifications are not sent in the next X minutes to avoid many parallel notifications in case many parallel conversations start.

Currently, an SMTP server is used that is freely provided by the internet service provider.

### 5.3.3 Data Import from an Optional Remote Database

There is a component "Optional Remote Database" on Figure 5.1. This means that the ADS framework can communicate with external databases.

This feature is implemented and used in a dialogue system (Alfred) that currently imports the new movie schedule into the ADS framework. This automated import takes place once a day.

ADS framework currently uses the HTTP protocol to connect to a remote database and import the movie schedule. The rest of the procedure (parsing the movie schedule to a suitable format) is rather task specific and probably not reusable.

# CHAPTER 6

# Application Issues and Evaluation

This chapter is about the application issues and evaluation. The intention is to show the process of getting the system to the state as it is currently in. This would provide information on further domain adaptations and also provides some insight on the flexibility of the ADS framework.

The process of resolving application issues includes evaluation from the point of developer and the evaluation from the point of the real users and the test users.

Most of the application issues (e.g. temporal resolution) were resolved during the analysis process and did not affect the overall implementation and deployment process. Yet, some of the interesting application issues (e.g. resolution of spelling errors) arose after testing of the first release of the framework. This chapter outlines these issues, the changes that were made to the framework to resolve these issues and the evaluation process.

## 6.1 Application Issues

The first release of the framework contained the core functionality, including the client side and the server side modules for basic chat interface. As the communication style seemed too limited the asynchronous communication was immediately added to the framework.

The next stage of the application process was the domain adaptation, starting from the knowledge engineering. This involved "character anticipation" (design of the domain character that the system had to become). This involved anticipating all the possible scenarios the system would get into and the information that the users might need.

As previously stated, the ADS framework is running as a base for two dialogue systems: Alfred and Zelda (Zelda, a virtual dental consultant, and Alfred, a provider of movie schedule information). These domains are rather different, which illustrates the flexibility and adaptability of the ADS framework.

After the knowledge had been added into the dialogue systems (Alfred and Zelda), the systems were ready for testing. A note to be made here is that the

number of rules in the knowledge base was about 20% of the current amount. This shows that the anticipation process is not as good as one would expect. Many unhandled issues seem to arise after deployment and after the real user testing. It is also important to note, that in building the knowledge base for Zelda, much assistance from a dental professionals was used. This external expert assistance could be inevitable for many expert domains.

The important feature that was added directly prior to the public testing was the interface for human assistance. The main reason for providing human assistance, was the fact that Zelda was built for a rather serious domain with real users. Without the human assistance, the number of disappointed users would be very high. For example, the public testing of Zelda without the human assistance would contain very high risk of doing serious damage to the reputation of the dental clinic that placed Zelda on its main internet page. So, the availability of human assistance in the framework is essential in the first public release. This also might be the reason why we don't see many dialogue systems in public use, because the frameworks usually lack the interface for human assistance.

## 6.2 Evaluation by Public Testing

The public testing quickly showed that spelling errors from the real users are very common. To resolve this issue, the usual solution would be to relax the pattern-matching process that would allow mistakes in the user input. Yet, this involves writing additional patterns (relaxed patterns) or a different parsing of regular expressions. None of these approaches are robust enough. Instead, ADS framework implements the support for resolving spelling errors – the process that executes immediately before the pattern-matching process (see Section 4.2).

The results in resolving spelling errors show that for Zelda the number of spelling errors corrected is 380 for the total of 4500 user utterances. That is 8.4% and it is a significant amount. This could well mean that 380 conversations would have failed without the speller.

The next issue that arose during the public testing was the word order problem. As the input from the real users had a free word order, the patterns had to be adjusted to resolve this issue.

As shown in Section 3.5 the approach that involved pattern modification was wrong as it resulted in pattern structures becoming more and more complex. This caused additional problems in the management of the knowledge base.

The correct solution to the word order problem was to add the word order resolution by allowing automated permutations.

As always, the improvements to the ADS framework (resolving of spelling errors and resolving word order problem) had to be tested. The best testing was to run a full check that was based on the previous input. The full test involved all the user input sentences that were automatically tested again, to see how

many of them would be resolved now as the framework has been improved. The tests showed that every improvement reduced the number of unanswered questions.

The similar testing was also carried out always after the knowledge base was improved. Yet, the improvement was rather small if only the rules were adjusted. For example, the increase in the number of rules by 20% (that means new rules were added into the knowledge base), the number of unanswered questions was reduced only by 10%.

It seems that at a certain point the increase in the number of rules gives a very little effect in the system quality. It shows that a certain small set of rules (approx 100 rules) are used very frequently and the rest of the rules are used very seldom.

The minor issue appeared with temporal queries that were necessary in Alfred but resulted in "*Not found.*" responses in Zelda. The solution was to allow this functionality of temporal resolution to be turned off in the ADS framework if not needed in a particular domain.

## 6.3 Evaluation by Test Users

Most of the user evaluation is based on Zelda (as Alfred was built mainly for illustrative reasons), yet many same aspects here apply for both of the dialogue systems.

The tests with the real users on Zelda show that the user disappointment level is quite low. The conversation logs show 20 utterances (from the total of 4500 utterances) that express disappointment of some level. Two of them expressed very high disappointment. This low rate is possible only because of the human assistance. Most of the Zelda users expressed gratitude at the end of the conversations. This could be influenced by the fact that the domain of the dialogue system was picked such that gave the users valuable information. If compared to Alfred, the movie information is not as valuable.

The real users were not questioned afterwards. The other tests were with test users and these users were questioned afterwards. However, the tests from the test users did not provide valuable input in the form of user utterances. The test users probably didn't have a dental issue to discuss with the system. So, they did not provide sentences that could come from the real users. Yet, they made some interesting remarks.

One of the complaints was the slow response rate. The test users were mostly very computer literate and fast in typing. So, they expected the same from the system – a fast response. Yet, the real users are not so fast and have expressed the opposite. So, this is the main reason for keeping the response timing a bit slower (the response timing is intentionally slow, it is not slow due to the computing speed or bad algorithms). The other reason is that the slow timing allows the human assistant to step in easily and give human assistance. It

takes about 30 seconds from receiving a sms-alert from the system, for the human assistant to get to the computer and start helping the system.

As the users are familiar with the timing from seeing the introducing sentences from the system being issued at this slow rate, they are still there waiting and the human assistant can assist. If the response timing would be faster, the users would consider the pause in the case of unanswered sentence too long and leave before the human assistant could provide assistance.

The solution to this slow response rate would be adjusting the response rate later on based on user typing speed.

A few minor remarks were on the appearance and usability of Alfred and Zelda. Most of the users gave very high rating on the easy chat interface that did not require any additional information or any additional actions from the user. The user was just able to enter the conversation and start typing right away. The users were satisfied that no registration, no menu selections, no setup of any kind was necessary, just a click on "start conversation" button and the chat interface was opened and ready to accept input.

Some of the test users suggested adding more "small talk" functionality. This was resolved by adding 20 rules for covering the small talk, such as answers to these "how are you?", "what have you been up to?", "how old are you?", "what are you?", etc.

## 6.4 Reducing the Amount of Human Assistance

As stated earlier, the goal of this research has always been maximizing the part of the conversation held with the program and minimizing the human intervention.

The additional features (correcting spelling errors, ignoring word order) and data collection with human assistance has helped to lower the percentage of human assisted answers.

It appears that in the early stages of testing while these new features were not available, 25% of answers were given by human operator in a conversation. After these new features were added and knowledge base was corrected, the percentage of human-assisted answers has decreased to 15%. We also must take into account the factor that the human-assistance was needed in some conversations, yet was not given as the human operator was not available. Therefore, it was necessary to check each conversation to see whether assistance was needed. This made the percentage go up a bit, up to 19%.

The ultimate goal is to have all answers given by the system and have 0% of human assistance. However, in most real life situations this goal will not be realistic. There will always be a small percentage of questions that need human assistance in answering. Therefore, this human-assisted approach seems a reasonable way to go in making these systems more available to public.

## 6.5 Word Count per Utterance

As one of the main features of the ADS framework is the asynchronous communication pattern (ACP), this chapter is concentrated on experiments looking into the benefits of the ACP. First of all the word count per utterance is examined, to determine whether using the ACP has any effect on making the users talk with more expressiveness (that is using more words per utterance and less single-word utterances).

The word count per utterance is an important issue in evaluating the mixed-initiative dialogue systems. This shows how expressive the users tend to be while talking to the dialogue system. It is not a good sign if the average number of words per utterance is very low. This means that the users have captured the keyword based approach of the system and are over-simplifying the conversation by using only single keywords. If the users use too many single-word utterances, the system is likely to fail in responding correctly [Liljenback, 2007].

Three dialogue systems were studied in evaluating the word count. One of the systems (*Teatriagent*) was a system with synchronous communication pattern which was not built in the ADS framework. The other two systems (*Alfred* and *Zelda*) were using asynchronous communication pattern and were implemented in the ADS framework.

Teatriagent and Alfred are very similar in domain – the first one gives information about theatre schedules and the second one gives information about the movie schedules. Zelda is a bit different in domain – a virtual dental consultant.

One of the aims of implementing the asynchronous communication pattern was to make the system more life-like, so that the users would not capture the keyword based approach of the system. We expected that in a life-like conversation pattern the users would try to be more expressive and use longer sentences while talking to the system. This would be beneficial as longer sentences contain more semantic information. We had a belief that when the system is not life-like in communication pattern, the users would tend to use just single keywords while talking to the system. The use of single keywords from the users is not a good sign and it makes it hard to provide appropriate answers.

Looking at the word count per utterance (see Table 6.1) shows that the systems with asynchronous communication pattern have higher numbers in word counts per utterance. It looks like the asynchronous communication pattern of the ADS framework has succeeded in making the users talk using more words.

Teatriagent has a large number (36.29%) of single-word utterances. Alfred and Zelda have less of these single-word utterances (accordingly 26.10% and 24.41%). This means, that users were more expressive while using Alfred and Zelda.

**Table 6.1:** Word count per utterance

| | Teatriagent | Aivo | Alfred | Zelda |
|---|---|---|---|---|
| average number of words in utterance | 2.66 | 3.20 | 3.26 | 4.71 |
| % of single-word utterances | 36.29% | 31.96% | 26.10% | 24.41% |
| % of utterances with more than 5 words | 6.71% | 14.31% | 12.12% | 24.04% |

There is also a system called *Aivo* that has been shown in the comparison (see Table 6.1). Aivo is also implemented in the ADS framework. However, Aivo is an exceptional system as it contains only WOZ conversations with asynchronous communication pattern. The number of single-word utterances is rather high in WOZ conversations (31.96%). This is probably due to the fact that the test users were not interested in asking information from Aivo. The recruited testers probably wanted to take the easy way out of the testing task. This resulted in using many single-word utterances.

The distribution of word count per utterance is shown in Figure 6.1.

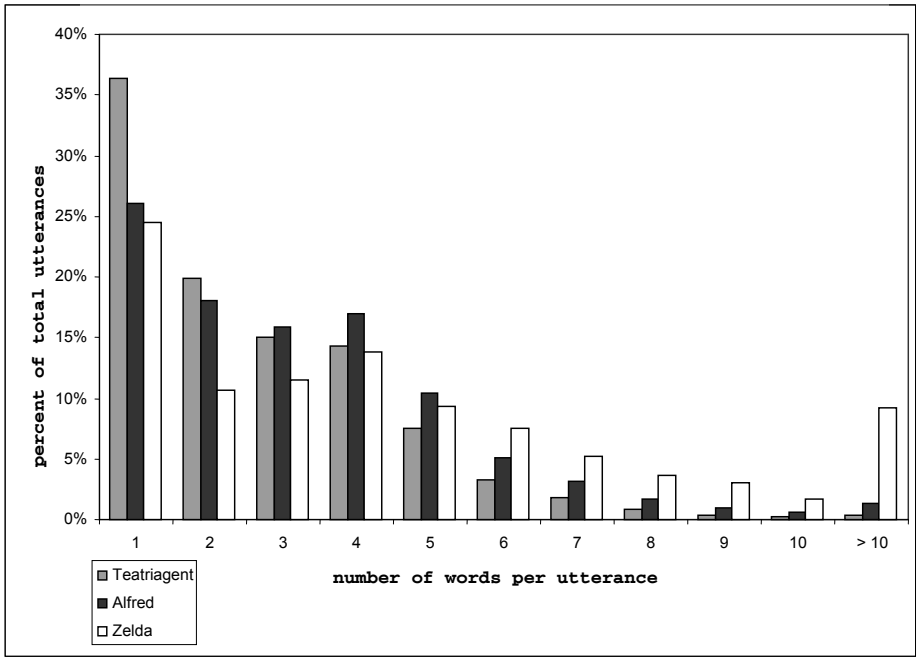



**Figure 6.1:** Word count per utterance.

The word count per utterance of Aivo is not shown in Figure 6.1 as it compares rather well to Alfred and the graph would become visually hard to capture.

There is one strange phenomenon to notice in Figure 6.1. The number of utterances with 4 words is higher than those with 3 words. This is true for Alfred and Zelda. There seems to be no meaningful explanation to this indication.

## 6.6 Wizard-of-Oz Experiments

The benefits of the asynchronous communication pattern in the process of WOZ data collection in text-based chat are discussed in Section 2.6.3.

It was stated that in the asynchronous communication pattern we have much better chances to trick the user into believing that the partner is still the computer, thus preventing the conversation from falling into the human-to-human communication pattern. The main reason being that asynchronous communication pattern helps to disguise the slower response rate of the human assistant.

There is no statistical data available to make a comparison to support this claim, i.e. to show how many times the slower response rates by the human assistants have caused the users to suspect that the system is assisted by a human.

However, it is a fact that 0% of WOZ conversations that were held with the ADS framework fell into the human-to-human communication pattern due to slower response rate problem.

There were some WOZ conversations with the ADS framework where the user discovered that the system is assisted by a human. However, all these problems were caused by the responses of the human assistant that were considered to be too intelligent for a system. Being too intelligent as a wizard, is another common give-away in WOZ conversations and there is no way to prevent this with the asynchronous communication pattern.

There is an alternative that is widely used to prevent the long pauses from causing suspicion. The task of WOZ data collection is set up so that all users are informed that the system is slow and makes longer pauses in answering.

# CHAPTER 7

# Future Work

This chapter lists a few ideas that will be implemented in the ADS framework in the near future.

## 7.1 Improving the Data Collection

Adding a new rule to the knowledge base is a subtask of knowledge engineering. The absence of a rule may be identified by analyzing the domain, possibly by browsing the dialogue corpora. In many cases, the lack of rules also appears during the WOZ process. So, the need to add a new rule usually arises from a new conversation topic or a new question that has not yet been handled by the system.

This approach is very inefficient, yet any automated ways to harvest knowledge are too much uncontrolled.

It would be reasonable to implement a semi-automatic approach in the ADS framework. This would involve supervised additions to the knowledge base during or after the WOZ process. The administrator is presented with the questions that were answered by WOZ and the administrator will use these question-answer pairs to design new rules. The question-answer pairs could be presented in Comma Separated Values (CSV) form and the words in the question could be presented as base forms. This would facilitate the process of adding new pattern-response pairs to the database.

An additional useful feature in managing the knowledge base would be an error-reporting solution. This reporting would provide information about duplicate rules (including duplication by word order), similar rules (by using Jaro Winkler) and syntax errors in rules.

## 7.2 Handling Data Update Requests by the User

A paper by Minock [Minock, 2006] opens the topic of natural language updates to databases. A database protocol to handle database updates of the *Insert-Delete-Modify* class is proposed and implemented. This protocol exploits

modern relational update facilities and constraints and structures update dialogues using DAMSL [Core and Allen 1997] dialogue acts Dialog. The protocol may be used with any natural language parser that maps to relational queries. So far the dialogue systems built on the ADS framework have been selecting data from the database (not inserting, deleting, updating), it would be interesting to experiment with an interface that would do updates to database.

## 7.3 Handling User Input in Multiple Passes

The user might issue one meaning in multiple passes, as in:

> User: I have a problem with my **tooth**.
> User: It **hurts**!

Or the user could issue even one sentence in multiple passes, as in:

> User: I would need to have my **tooth**, how to say it …
> User: **extracted** or **removed**.

In this case the match is not found, as none of the inputs did contain suitable patterns: `[tooth hurt; tooth extracted|removed].`

Yet, there is enough information available to find a meaning in such input and to provide an answer.

The pattern search in the user input currently looks at every specific input pass. The solution would be to extend the pattern search to multiple passes. We would need to concatenate the consequtive unanswered input sentences and try to search for a pattern again.

# CHAPTER 8

# Conclusions

This thesis presents the ADS framework – a collection of integrated modules (including several NLP modules) that can be used in developing text-based natural language dialogue systems. The ADS framework is currently tailored for Estonian language, yet most of its features and modules are easily transferable to English language. The dialogue systems based on the ADS framework mimic the natural interaction between people better than the models used so far.

The ADS framework provides a hybrid approach – "a human assisted chat system" that allows a single human agent to handle a number of simultaneous chat sessions by having an AI-engine handle the bulk of common, repeat questions. The AI-engine will allow the human agent to focus his or her attention on the few chat sessions needing unique service and will effectively lower the cost of supporting chat sessions. The server-side technology of the ADS framework uses an AI-engine as well as a live agent backend interface for a site to deliver live-agent experience without the customer having to know whether the answer is from the AI-engine or from the human agent.

This approach allows us to put these dialogue systems into practical use and avoid user disappointment. Although, the dialogue systems developed in the ADS Framework, can be assisted by a human, still the goal of this research has always been maximizing the AI-participation in the conversation and minimizing the human intervention.

The contributions of the thesis are listed as the features of the ADS framework, including:

a) Easy and compact representation of knowledge, so that the domain adaptation and knowledge base engineering would contain a minimal amount of programming effort. The knowledge is represented as a set of pattern-response pairs. The system also includes pattern-function pairs to represent procedural knowledge. The patterns are expressed as regular expressions with added support for free word order problem.
b) Asynchronous turn-taking strategy, so that both parties (human and computer) can provide input at any given moment and can take any number of sequential turns without waiting for the other party to acknowledge each turn.

c) AI-assisted live agent chat, so that the unanswered questions can be handled by an optional human operator.
d) Robust language analysis, so that the misspellings in the user input are corrected by the system. The language analysis also includes the stemming (the process of reducing a word to its root word), to ease the pattern creation in knowledge engineering.
e) Separation of declarative domain knowledge and procedural code. The domain specific knowledge and temporal constraints are separated from the central dialogue management.
f) A flexible and modular design for rapid development of mixed-initiative dialogue systems with a web-based interface.
g) A web-based conversation interface with optional speech synthesis.
h) A language independent solution for the word-order problem, thus allowing skipping the syntactic analysis and optionally ignoring the word-order problem in the knowledge engineering process. This is essential for languages with relatively free word order (such as Estonian).
i) A collection of temporal constraints for Estonian temporal normalization.

# Bibliography


[AIML, 2009] AIML (2009). *Artificial Intelligence Markup Language.* Retrieved Aug 1, 2009 from http://www.alicebot.org/aiml.html

[Allaire, 2002] Allaire, J. (2002) Macromedia Flash MX – A next-generation rich client. Technical report, Macromedia.

[Allen et al, 2001] Allen, J., Ferguson, G. and Stent, A. (2001). An architecture for more realistic conversational systems. In *IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces*, Santa Fe, New Mexico, United States, pages 1–8. ACM Press.

[Alumäe, 2006] Alumäe, T. (2006). *Methods for Estonian large vocabulary speech recognition*. Ph.D. Thesis. Tallinn University of Technology. TUT Press.

[Arnold and Gosling, 1996] Arnold, K. and Gosling, J. (1996). The Java Programming Language. Reading, Mass.: Addison-Wesley.

[Atkinson and Suraski 2003] Atkinson, L. and Suraski, Z. (2003). Core PHP Programming, Third Edition, Prentice Hall Professional Technical Reference.

[Black, 2005] Black, P. E. (2005). Dictionary of Algorithms and Data Structures. Information Technology Laboratory, National Institute of Standards and Technology (NIST).

[Blaylock et al., 2002] Blaylock, N., Allen, J., Ferguson, G. (2002). Synchronization in an asynchronous agent-based architecture for dialogue systems. In: *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, pages 1–10, Philadelphia, Pennsylvania.

[Bohus and Rudnicky, 2003] Bohus, D. and Rudnicky, A. (2003). RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda. In: *Proceedings of Eurospeech*.

[Bohus et al., 2007] Bohus, D., Raux, A., Harris, T., Eskenazi, M. and Rudnicky, A., (2007). Olympus: an open-source framework for conversational spoken language interface research. In: *Proceedings of HLT-2007*. Rochester, NY.

[Cimiano et al., 2007] Cimiano, P., Haase, P. and Heizmann, J. (2007). Porting natural language interfaces between domains – a case study with the ORAKEL system. In: *Proceedings of the International Conference on Intelligent User Interfaces*, pages 180–189.

[Cohen et al., 2003] Cohen, W. W., Ravikumar, P. and Fienberg, S. E. (2003). A comparison of string metrics for matching names and records. In: *Proceedings of KDD-2003 Workshop on Data Cleaning and Object Consolidation*.

[Core and Allen 1997] Core, M. and Allen, J. (1997). Coding dialogs with the DAMSL annotation scheme. In: *AAAI Fall Symposium on Communicative Action in Humans and Machines*, Boston, MA.

[Crockford, 2008] Crockford, D. (2008). JavaScript: The Good Parts, O'Reilly Media, Inc.

[CSLU Toolkit, 2009] *CSLU Toolkit*. (2009). Retrieved Aug 1, 2009 from http://cslu.cse.ogi.edu/toolkit/

[Dzikovska et al., 2003] Dzikovska, M.O., Allen, J.F. and Swift, M.D. (2003). Integrating linguistic and domain knowledge for spoken dialog systems in multiple domains. In: *Proc. of IJCAI-03 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, Acapulco, Mexico.

[Eichorn, 2006] Eichorn, J. (2006). Understanding AJAX: Using JavaScript to Create Rich Internet Applications, Prentice Hall PTR, Upper Saddle River, NJ.

[Elder, 2004] Elder, M. (2004). *Preparing a data source for a natural language query*. United States Patent Application No 20050043940.

[Fielding et al., 1997] Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and Berners-Lee, T. (1997). Hypertext Transfer Protocol – HTTP/1.1, RFC Editor.

[Jokinen et al., 2002] Jokinen, K., Kerminen, A., Kaipainen, M., Jauhiainen, T., Wilcock, G., Turunen, M., Hakulinen, J., Kuusisto, J. and Lagus, K. (2002). Adaptive dialogue systems – interaction with interact. In: *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*.

[Jurafsky and Martin, 2000] Jurafsky, D. and Martin, J. H. (2000), Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Prentice-Hall, Upper Saddle River, NJ.

[Kaalep and Vaino, 2001] Kaalep, H.-J. and Vaino, T. (2001). Complete morphological analysis in the linguist's toolbox. In: *Congressus Nonus Internationalis Fenno-Ugristarum Pars V*, pages 9–16, Tartu, Estonia.

[Lie and Bos, 1997] Lie, H.W. and Bos, B. (1997). Cascading style sheets: designing for the Web, Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

[Liljenback, 2007] Liljenback, M. E. (2007). ContextQA: Experiments in Interactive Restricted-Domain Question Answering, MSc. in CS Thesis, San Diego University, 2007.

[Loney, 2004] Loney, K. (2004). Oracle Database 10g: The Complete Reference, Osborne. McGraw-Hill, New York.

[Lucas, 2000] Lucas, B. (2000). VoiceXML for Web-based distributed conversational applications, Communications of the ACM, Vol. 43, pages 53–57.

[Meister et al., 2003] Meister, E., Lasn, J. and Meister, J. (2003). SpeechDat-like Estonian database. In: *Text, Speech and Dialogue : 6th International Conference, TSD 2003*, Czech Republic, September 8–12, 2003 / Eds. Matoušek [et al.]. Berlin [etc.] : Springer, Lecture Notes in Artificial Intelligence, Vol. 2807, pages 412–417.

[Mihkla et al., 1999] Mihkla, M., Eek, A. and Meister, E. (1999). Text-to-Speech Synthesis of Estonian. In: *Proceedings of the 6th European Conference on Speech Communication and Technology*, Budapest, Vol. 5, pages 2095–2098.

[Minock, 2006] Minock, M. (2006). Natural language updates to databases through dialogue. In: *Proceedings of Applications of Natural Language to Data Bases (NLDB)*, pages 203–208, Klagenfurt, Austria.

[Misu et al., 2010] Misu, T., Ohtake, K., Hori, C., Kashioka, H., Kawai, H. and Nakamura, S. (2010). Construction and Experiment of a Spoken Consulting Dialogue System. In Proc. IWSDS.

[Popescu et al., 2003] Popescu, A.-M., Etzioni, O. and Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In: *Proceedings of the International Conference on Intelligent User Interfaces*, pages 149–157, Miami, USA.

[Postel, 1982] Postel, J. (1982). Simple Mail Transfer Protocol, RFC Editor.
[Raggett et al, 1999] Raggett, D., Hors, A. L. and Jacobs, I. (1999). HTML 4.01 Specification. W3C Recommendation. Retrieved May 12, 2011 from http://www.w3.org/TR/html4/
[Raynor, 1999] Raynor, W. (1999). The International Dictionary of Artificial Intelligence, USA: The Glenlake Publishing Company Ltd.
[Rieser and Lemon, 2008] Rieser, V. and Lemon, O. (2008). Learning Effective Multimodal Dialogue Strategies from Wizardof-Oz data: Bootstrapping and Evaluation, In: *Association for Computational Linguistics (ACL)*, Columbus, OH, USA, pp. 638–646.
[Sajja and Akerkar, 2010] Sajja, P.S. and Akerkar, R. (2010). Knowledge-Based Systems for Development, Advanced Knowledge Based Sytem: Model, Application & Research, Vol. 1, 2010, pages 1–3.
[Semantra, 2009] Semantra, (2009). *Semantra Technology Overview*. Retrieved Aug 1, 2009 from http://www.semantra.com/solutions/technology.cfm
[Stenchikova and Stent, 2007] S. Stenchikova and A. Stent. (2007). Measuring adaptation between dialogs. In: *Proceedings of the 8th SIGdial Workshop on Discourse and Dialogue*, Antwerp, Belgium.
[Sutrop, 2004] Sutrop, U. (2004). *Estonian language*. Retrieved Aug 1, 2009 from http://www.einst.ee/failid/eestikeel.web_1.pdf
[Sutton et al., 1998] Sutton, S., Cole, R., De Villiers, J., Schalkwyk, J., Vermeulen, P., Macon, M., Yan, Y., Kaiser, E., Rundle, B., Shobaki, K., Hosom, J.P., Kain, A., Wouters, J., Massaro, D. and Cohen, M. (1998). Universal speech tools: The CSLU toolkit. In: *Proceedings of the 5th International Conference on Spoken Language Processing* (ICSLP'98, Sydney, Australia). ICSLP, 3221–3224.
[Toney et al., 2008] Toney, D., Rosset, S., Max, A., Galibert, O. and Bilinski, E. (2008). An Evaluation of Spoken and Textual Interaction in the RITEL Interactive Question Answering System. In: *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco.
[Treumuth et al., 2006] Treumuth, M., Alumäe, T. and Meister, E. (2006). A Natural Language Interface to a Theater Information Database. In: *Proceedings of the 5th Slovenian and 1st International Language Technologies Conference 2006 (IS-LTC 2006)*, pages 27–30.
[Treumuth, 2008] Treumuth, M. (2008). Normalization of Temporal Information in Estonian. In: *Proceedings of the 11th international conference on Text, Speech and Dialogue*. Brno, Czech Republic.
[VoiceXML, 2009] VoiceXML, (2009). *VoiceXML specification*. Retrieved Aug 1, 2009 from http://www.voicexml.org
[Vrajitoru, 2006] Vrajitoru, D. (2006). NPCs and Chatterbots with Personality and Emotional Response. In: *Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games (CIG06)*, pages 142–147.
[Widenius et al., 2002] Widenius, M., DuBois, P. and Axmark, D. (2002). Mysql Reference Manual, O'Reilly & Associates, Inc., Sebastopol, CA.
[Wilson, 2000] Wilson, B. (2000). The Natural Language Processing Dictionary. Retrieved May 12, 2011 from http://www.cse.unsw.edu.au/~billw/nlpdict.html

# Kokkuvõte (Summary in Estonian)

## Asünkroonsete dialoogsüsteemide raamistik: mõisted, probleemid ja kavandamise aspektid

Käesolevas doktoritöös realiseeriti nn. *asünkroonsete dialoogsüsteemide (ADS)* raamistik – tarkvara, mida saab kasutada tekstipõhiste, kasutajaga loomulikus keeles üle Interneti suhtlevate dialoogsüsteemide loomisel. Eeskätt on arvestatud eestikeelse suhtlusega, kuid kuna süsteem on modulaarne ja enamik mooduleid on püütud teha keelest sõltumatuks, siis on raamistik – erinevalt olemasolevaist – suhteliselt hõlpsasti ülekantav ka teistele keeltele. Raamistikus realiseeritud suhtlusmudel jäljendab loomulikku inimestevahelist suhtlust paremini kui seni kasutusel olnud mudelid.

Dialoogsüsteemide laiema praktilise kasutuse üheks peamiseks takistuseks on asjaolu, et süsteemid ei ole veel piisavalt head, et garanteerida kasutaja rahulolu. Ka kõige parema dialoogsüsteemi puhul on ebareaalne eeldada, et süsteem suudab korrektselt lahendada 100% kasutaja pöördumistest.

ADS raamistik võtab kasutusele kombineeritud lähenemise: inimabiga dialoogsüsteemid, kus üksikutele süsteemi poolt vastamata jäetud küsimustele vastab reaalajas inimoperaator. Seejuures teeb ta seda nõnda, et kasutaja jaoks ei ole inimoperaatori või arvuti vastused eristatavad.

Selline lahendus on analoogiline masintõlke valdkonnas kasutatava lähenemisega. Ka masintõlge ei suuda käesoleval hetkel pakkuda 100% korrektsust ning inimene peab kas tõlkeprotsessi sekkuma või tõlgitud teksti redigeerima.

Inimabiga dialoogsüsteeme on turvalisem pakkuda praktilisse kasutusse ning tulenev lisaväärtus on arvestatav nii kasutajaile kui ka arendajaile. Näiteks infotelefonikõned saab nüüd asendada suhtlusega Internetis ja seejuures delegeerida suures osas vastamine masinale. Kümnete samaaegsete vestluste pidamiseks piisab ühestainsast operaatorist, kes aitab vaid üksikuid vestlussessioone, kus arvuti vastamisel hädas on. Ülejäänud juhtudega, kus vestlus kulgeb tavapärast rada, suudab tegeleda masin. Seeläbi saab kokku hoida inimoperaatorite aega ja pakkuda suhtlemisvõimalust suuremale kasutajaskonnale.

Arendaja seisukohalt on aga lisaväärtuseks reaalse kasutuse käigus tekkivate andmete suur hulk. Dialoogsüsteemi varjatud abistamisel saab koguda vestlusi ja nende põhjal hiljem laiendada süsteemi teadmisi. Dialoogsüsteemide arendajana on autor märganud, et reaalse kasutuse käigus tekkivad andmed on arendajale tunduvalt väärtuslikumad kui testijate poolt tekitatud kunstlikud stsenaariumid, mida reaalses elus enamasti ei esine.

ADS raamistikku realiseerides ei olnud siiski põhieesmärk inimabi liidese loomine. Laiem eesmärk on jätkuvalt uurida inimese ja arvuti vahelises suhtluses tekkivaid probleeme ning töötada välja lahendusi nende probleemide ületamiseks. Nõnda pidevalt

parandades konkreetset dialoogsüsteemi ja/või ADS raamistikku, on inimabi osakaal vestlustes üha langenud.

ADS raamistiku üheks peamiseks iseärasuseks, nagu ütleb ka raamistiku nimi, on asünkroonse suhtlusmudeli kasutuselevõtt. Kui varem loodud dialoogsüsteemides oli süsteemi voorude arv jäigas vastavuses kasutaja voorude arvuga, siis uus vestlusmudel teeb suhtluse loomulikumaks: dialoogsüsteem ei pea enam ootama kasutajalt sisendit, püsides vaikides ooterežiimis, vaid saab vabalt valitud hetkel ise vooru võtta. Töös näidatakse, et asünkroonse vestlusmudeli kasutuselevõtt parandas ka kasutaja vestlusstiili, mille üheks tulemuseks oli ühesõnaliste lausungite arvu vähenemine. Suurem sõnade arv lausungis parandab aga lause mõistmist ja tõstab süsteemi korrektse vastuse tõenäosust.

Lisaks eeltoodule, on ADS raamistiku mõned olulisemad panused:

- keelest sõltumatu lahendus õigekirja kontrollimiseks ja kasutaja sisendis leiduvate võimalike kirjavigade automaatseks parandamiseks;
- keelest sõltumatu lahendus sõnade järjekorra ignoreerimiseks, mis ühelt poolt annab võimaluse loobuda süntaktilisest analüüsist ning teisalt teeb lihtsamaks teadmusbaasi täiendamise;
- komplekt eestikeelseid ajaväljendeid ja nende esitus formaalsete kitsendustena, mida saab kasutada eestikeelsete ajaväljendite normaliseerimiseks info otsingul.

# Appendix A

# Technical specifications

**1. List of acronyms (and their references if applicable):**

| | |
|---|---|
| ADS | Asynchronous Dialogue System |
| AJAX | Asynchronous JavaScript and XML<br>http://www.w3.org/TR/XMLHttpRequest/<br>Retrieved 07.04.2011. |
| CSS | Cascading Style Sheets<br>http://www.w3.org/TR/CSS21/<br>Retrieved 07.04.2011. |
| GUI | Graphical User Interface |
| HTML | HyperText Markup Language<br>http://www.w3.org/TR/1999/REC-html401-19991224/<br>Retrieved 07.04.2011. |
| XHTML | eXtensible HyperText Markup Language<br>http://www.w3.org/TR/2010/REC-xhtml11-20101123/<br>Retrieved 07.04.2011. |
| HTTP | HyperText Transfer Protocol<br>http://tools.ietf.org/html/rfc2616<br>Retrieved 07.04.2011. |
| PHP | PHP: Hypertext Preprocessor (scripting language)<br>http://www.php.net/<br>Retrieved 07.04.2011. |
| PL/SQL | Procedural Structured Query Language<br>http://www.oracle.com/<br>Retrieved 07.04.2011. |

SMTP Simple Mail Transfer Protocol
http://tools.ietf.org/html/rfc5321
Retrieved 07.04.2011.

SQL Structured Query Language
http://www.oracle.com/
Retrieved 07.04.2011.

XML Extensible Markup Language
http://www.w3.org/TR/REC-xml/
Retrieved 07.04.2011.

## 2. A brief summary of technical details

The ADS framework is a collection of integrated modules that can be used in developing text-based natural language dialogue systems. The ADS framework is very much a server-centric model where client-side code is kept to a minimum. The client-side logic is written in JavaScript [Crockford, 2008] (AJAX). The server-side logic is written in PHP and Oracle PL/SQL. Flash plugin [Allaire, 2002] is used for sound support (audio playback of optional speech synthesis).

AJAX (asynchronous JavaScript and XML) is a group of interrelated web development techniques used for creating interactive web applications or rich Internet applications. With AJAX, the dialogue system can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. Data is retrieved using the XMLHttpRequest object.

XMLHttpRequest is an API that is used by JavaScript to transfer XML and other text data between a web server and a browser. The data returned from XMLHttpRequest calls is provided by the back-end database. Using Ajax, the dialogue system can request only the content that needs to be updated, thus reducing the bandwidth usage.

The layout of chat client is fully customizable by using CSS (Cascaded Style Sheets). XHTML is used for the content-type.

No setup is needed on the client machine. The client works within the browser. The system is successfully tested with common web browsers (most recent versions of Firefox, Opera, Internet Explorer and Chrome). No browser plugins are required for main functionality. Optional Flash 9.0 is required for speech synthesis.

# Appendix B

# Estonian language: noun cases and verb inflections

### 1. Estonian cases [Sutrop, 2004]

| Case | Example | Meaning |
|---|---|---|
| ***Grammatical cases*** | | |
| 1. Nominative | ilus maja | (a) beautiful house |
| 2. Genitive | ilusa maja | of a beautiful house; a beautiful house (as a total object) |
| 3. Partitive | ilusa-t maja | a beautiful house (as a partial object) |
| ***Semantic cases*** | | |
| *Interior local cases* | | |
| 4. Illative | ilusa-sse maja-sse (ilusasse majja) | into a beautiful house |
| 5. Inessive | ilusa-s maja-s | in a beautiful house |
| 6. Elative | ilusa-st maja-st | from a beautiful house |
| *Exterior local cases* | | |
| 7. Allative | ilusa-le maja-le | onto a beautiful house |
| 8. Adessive | ilusa-l maja-l | on a beautiful house |
| 9. Ablative | ilusa-lt maja-lt | from a beautiful house |
| *Other cases* | | |
| 10. Translative | ilusa-ks maja-ks | [to turn] (in)to a beautiful house |
| 11. Terminative | ilusa maja-ni | up to a beautiful house |
| 12. Essive | ilusa maja-na | as a beautiful house |
| 13. Abessive | ilusa maja-ta | without a beautiful house |
| 14. Comitative | ilusa maja-ga | with a beautiful house |

**2. Estonian verb inflections [Alumäe, 2006]**

| Plurality | Person | Example | Meaning |
|---|---|---|---|
| Singular | I | (ma) armasta-n | I love |
| | II | (sa) armasta-d | you love (sg.) |
| | III | (ta) armasta-b | he/she/it loves |
| Plural | I | (me) armasta-me | we love |
| | II | (te) armasta-te | you love (pl.) |
| | III | (nad) armasta-va-d | they love |

# Appendix C

# Excerpts from the knowledge bases

### 1. Examples of temporal constraints used in temporal resolution

| Temporal expression in Estonian | Constraint in SQL |
|---|---|
| (nädalavahetus\|nädalalõpp) | trim(to_char(kuupaev, 'DAY')) = 'SATURDAY' or trim(to_char(kuupaev, 'DAY')) = 'SUNDAY' |
| hommik | to_char(kuupaev, 'hh24:mi') between '06:00' and '12:00' |
| lõuna | to_char(kuupaev, 'hh24:mi') between '12:01' and '14:00' |
| õhtu | to_char(kuupaev, 'hh24:mi') between '17:01' and '23:00' |
| öö | (to_char(kuupaev, 'hh24:mi') between '23:01' and '23:59' or to_char(kuupaev, 'hh24:mi') between '00:00' and '05:59') |
| üleeile | trunc(kuupaev) = trunc(sysdate)-2 |
| eile | trunc(kuupaev) = trunc(sysdate)-1 |
| täna | trunc(kuupaev) = trunc(sysdate) |
| homme | trunc(kuupaev) = trunc(sysdate)+1 |
| ülehomme | trunc(kuupaev) = trunc(sysdate)+2 |
| esmaspäev | trim(to_char(kuupaev, 'DAY')) = 'MONDAY' |
| teisipäev | trim(to_char(kuupaev, 'DAY')) = 'TUESDAY' |
| ... | |
| (jaanuar\|näärikuu) | to_char(kuupaev, 'mm') = 1 |
| (veebruar\|küünlakuu) | to_char(kuupaev, 'mm') = 2 |
| (märts\|paastukuu) | to_char(kuupaev, 'mm') = 3 |
| (aprill\|jürikuu) | to_char(kuupaev, 'mm') = 4 |
| (mai(l\|ni\|le)?\|lehekuu) | to_char(kuupaev, 'mm') = 5 |
| (juuni\|jaanikuu) | to_char(kuupaev, 'mm') = 6 |
| ... | |
| suvi | to_char(kuupaev, 'mm') in (6, 7, 8) |
| sügis | to_char(kuupaev, 'mm') in (9, 10, 11) |
| talv | to_char(kuupaev, 'mm') in (12, 1, 2) |
| kevad | to_char(kuupaev, 'mm') in (3, 4, 5) |
| (hommikul \|õhtul \|öösel \|lõuna ajal )?kell (\d{1,2}\:\d{2}) | to_char(kuupaev, 'hh24:mi') = lpad('\2', 5, '0') |
| (hommikul \|õhtul \|öösel \|lõuna ajal )?kell (\d{1,2}) | to_char(kuupaev, 'hh24') = lpad('\2', 2, '0') |
| (mõne\|kolme) tunni pärast | kuupaev between sysdate+2.5/24 and sysdate+3.5/24 |
| ((paari\|kahe) tunni eest\|(paar\|kaks) tundi tagasi) | kuupaev between sysdate-2.5/24 and sysdate-1.5/24 |
| (\d{1,2})\.?\s?(jaanuar\|näärikuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.01' |

| Temporal expression in Estonian | Constraint in SQL |
|---|---|
| (\d{1,2})\.?\s?(veebruar\|küünla-kuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.02' |
| (\d{1,2})\.?\s?(märts\|paastukuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.03' |
| (\d{1,2})\.?\s?(aprill\|jürikuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.04' |
| (\d{1,2})\.?\s?(mai(l\|ni\|le)?\| lehekuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.05' |
| (\d{1,2})\.?\s?(juuni\|jaanikuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.06' |
| (\d{1,2})\.?\s?(juuli\|heinakuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.07' |
| (\d{1,2})\.?\s?(august\|lõikuskuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.08' |
| (\d{1,2})\.?\s?(september\|mihkli-kuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.09' |
| (\d{1,2})\.?\s?(oktoober\|viina-kuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.10' |
| (\d{1,2})\.?\s?(november\|talve-kuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.11' |
| (\d{1,2})\.?\s?(detsember\|jõulu-kuu) | to_char(kuupaev, 'dd.mm') = lpad('\1', 2, '0') \|\| '.12' |
| ((mõne\|kolme) tunni eest\|(mõni\|kolm) tund tagasi) | kuupaev between sysdate-3.5/24 and sysdate-2.5/24 |
| (paari\|kahe) tunni pärast | kuupaev between sysdate+1.5/24 and sysdate+2.5/24 |
| (tunni (aja )?pärast\|ühe tunni pärast) | kuupaev between sysdate+1/24 and sysdate+1.5/24 |
| (tunni (aja )?eest\|tund (aega )?tagasi) | kuupaev between sysdate-1.5/24 and sysdate-1/24 |
| jõulud | to_char (kuupaev, 'dd.mm') in ('24.12', '25.12', '26.12') |
| (näärid\|uusaasta) | to_char (kuupaev, 'dd.mm') = '01.01' |
| (valentinipäev\|sõbrapäev) | to_char (kuupaev, 'dd.mm') = '14.02' |
| (aprillipäev\|naljapäev) | to_char (kuupaev, 'dd.mm') = '01.04' |
| (volbripäev\|maipüha\|töörahvapüha\| kevadpüha) | to_char (kuupaev, 'dd.mm') = '01.05' |
| (esimene koolipäev\|tarkusepäev) | to_char (kuupaev, 'dd.mm') = '01.09' |
| (jõululaupäev\|jõuluöö\|jõuluõhtu) | to_char (kuupaev, 'dd.mm') = '24.12' |
| esimene jõulupüha | to_char (kuupaev, 'dd.mm') = '25.12' |
| teine jõulupüha | to_char (kuupaev, 'dd.mm') = '26.12' |
| ... | |
| pärastlõuna | to_char(kuupaev, 'hh24:mi') between '14:01' and '17:00' |
| ... | |
| ... | |
| ... | |

## 2. Examples of rules from the knowledge base of Zelda

The meaning of the column "X" is "Ignore word order" with values Y and N (Yes/No). If value = Y then the word order can be ignored while matching this pattern.

The sequence "…" in regular expressions denotes the option for intermediate words and is replaced during the parsing with the sequence "`( .*)?`".

| Pattern | Response | State | X |
|---|---|---|---|
| aadress\|tänav\|asukoht\|kaart\|kuhu\|kontakt\|(kohale tulema)\|(kus ... asuma) | meie asukoha leiate siit [vaata kontaktandmeid] | | Y |
| aeg ... olema\|kellaaeg\|kell\|millal\|(vastu võtma)\|päeviti\|päevadel\|vastuvõtt\|(millal\|kuna) ... (avatud\|lahti) | Dr. **** vastuvõtuajad on: E 9-17, T 14-20 ja N 9-17, aga saab ka muu aja kokku leppida | | Y |
| anesteesia\|anestesioloog\|tuimestus\|tuimestama\|narkoos\|valutu\|üldnarkoos\|magama ... (panema\|pand\|panna\|panek\|panemine) | üldnarkoosi meil ei tehta, tehakse paikne tuimestus | NARKOOS | Y |
| arsti ... juurde | aga siis tasuks tulla vastuvõtule | | N |
| (asendama\|asendamine\|asemele\|puuduolev\|puudu\|puuduma\|puuduv) | kaotatud hammaste asendamiseks on kaks varianti: sild või implantaat | IMPLANTAAT | N |
| breket.*\|puseriti\|risti\|risthambumus | ortodontiaga (sh breketitega) meie ei tegele, vaadake näiteks [www.****.ee] | | N |
| büügel\|büügelprotees | büügel on suust eemaldatav protees, mis koosneb metallkarkassist ja plastmassist hammastest | | N |
| diagnoos\|diagnoosima | diagnoosi saame panna esmasel visiidil, kuigi vajalik oleks ka panoraamülesvõte (röntgen) | | N |
| garantii | garantiid reguleerib tarbijakaitseseadus, põletiku ravis puudub garantii, materjalidele on garantii | | N |
| gingiviit\|igemepõletik\|paradondiit\|parodontiit\|parodontoos\|igemehaigus | gingiviit on igeme põletik ja parodontiit on hammast ümbritsevate kudede krooniline haigus | PARODONTIIT | N |
| haiget | ei saa väga haiget, on ju tuimestus ja kaasaegsed vahendid | | N |
| (halb\|ebameeldiv\|vastik) ... (hingeõhk\|hais) | halb hingeõhk võib olla igemepõletikuga seotud | | Y |
| (halb\|paha) olema | põletikus hambad kahjustavad üldtervist oluliselt | | Y |
| hambakaunistus\|briljant\|kaunistus | paigaldame hambakaunistusi | | N |
| hambakivi\|kivi\|katt\|hambakatt | hambakivi on hambale ladestunud kõva katt, mis tuleks kindlasti arstil eemaldada | HAMBAKIVI | N |
| hambapasta\|pasta\|suuvesi\|loputus\|loputusvahend\|hambahari\|hari | hambaharja, -pasta ja suuvee valik on personaalne ja määratakse esmasel visiidil | KODUNE HOOLDUS | N |

| Pattern | Response | State | X |
|---|---|---|---|
| hambapesu\|(hammas ... (pesema\|pesemine\|harjamine\|harjama)) | hammaste pesemine aitab ära hoida pindmise põletiku e. gingiviidi teket. luupõletikku e. parodontiiti ainult hambapesuga ravida ei saa | KODUNE HOOLDUS | Y |
| hambaravi | pakume hambaravi, igemeravi, proteesimist ja implantaate | | N |
| hammas ... (viltu\|viltune) | ortodontiaga (sh breketitega) meie ei tegele, vaadake näiteks [www.****.ee] | | Y |
| helistama\|helistamine\|telefon\|registreeruma\|registreerima | helistage julgesti Dr. **** numbril **** | | N |
| hügienist\|suuhügienist | korra aastas võiks käia hügienisti vastuvõtul, kus eemaldatakse katt ning hambakivi | | N |
| hüvasti\|(head aega)\|nägemist | nägemist | | N |
| ibumetin\|paracetamol\|paracetamool\|paratsetamol\|paratsetamool\|ibuprofen\|ibuprofeen\|ibumax | tegemist on jah tuntud valuvaigistiga, millega võib valu ajutiselt maha võtta | RAVIM | N |
| igemeravi | igemeravi on hammast ümbritsevate kudede ravi | IGEMERAVI | N |
| implantaat ... (breket\|breketid\|breketravi\|breketite) | implantaat tuleks paigaldada alles pärast breketravi lõppu | | Y |
| implantaat\|implantaaž\|implantatsioon\|tehisjuur\|kruvi\|tehishammas\|valehambad\|kunsthammas\|hambaimplantaat\|(luu ... sisse) | põhjalikum info on siin: [loe implantaatidest] | IMPLANTAAT | N |
| internet ... (aeg\|aega)\|vaba ... (aeg\|aega)\|kuidas ... (aeg\|aega)\|((aeg\|aega) ... kinni)\|kirja\|kirjutama\|kiri | interneti kaudu saab aega küsida siit [saada kiri] | | Y |
| juureravi | juureravi pakume, täpsema info saame anda esmasel visiidil | | N |
| järelmaks\|järelmaksuvõimalus\|((maksma\|tasuma) ... ([^]*haaval\|kaupa))\|graafik\|osamaks\|osamakse\|((maksma\|tasuma) ... (osas\|osaliselt)) | teatud osas on võimalik järelmaks | JÄRELMAKS | Y |
| järjekord | järjekorrad ei ole meil üldjuhul pikad | JÄRJEKORD | N |
| kaaries\|auk\|hambaauk | kaaries on hamba kõvakudede kroonilise kuluga haigus, mille tagajärjel tekib hambasse auk | | N |
| (kaotama\|kaotatud\|kukkuma\|(tulema\|tuli) ... (välja\|ära)\|lööma\|looma\|langema\|löödi\|lõi\|lõid) ... hammas | kaotatud hammaste asendamiseks on kaks varianti: sild või implantaat | IMPLANTAAT | Y |
| kartma\|hirm\|hirmunud\|hirmuma\|kartus\|pelgama\|kartmine\|julge\|hirmutama\|julgus\|julgema | ärge kartke, teie probleemi suhtutakse mõistvalt ja professionaalselt | | N |
| kestvus\|kestma | ravi kestvus on individuaalne ja selgub esmasel visiidil | | N |
| kirurg\|doktor\|arst\|hambaarst | Dr. **** number on **** | | N |
| kloorheksidiin\|loputusvedelik\|loputama\|soolvesi\|lahus | kloorheksidiini kasutatakse profülaktilise vahendina (loputusvedelik) | RAVIM | N |
| (kodu\|kodune\|ise) ... (ravima\|ravi\|hoolitsema\|hoolitsus\|hooldus\|tegema) | koduse ravi ja hoolduse saab määrata arst | KODUNE HOOLDUS | Y |

| Pattern | Response | State | X |
|---|---|---|---|
| krediit\|krediitkaart\|pangakaart\|kaardimakse\|maksma ... kaardiga | kahjuks meie juures saab maksta vaid sularahas või ülekandega | | Y |
| krigistamine\|krigistama | hammaste krigistamine on halb harjumus | | N |
| laminaat | portselanist või plastikust valmistatavad laminaadid on õhukesed "koorikud", mis liimitakse hamba esipinnale | | N |
| loksuma\|liikuma\|loksumine\|liikumine\|liikuvus\|logisema | hammaste liikuvus (loksumine) on ohu märk, tuleks tulla visiidile | | N |
| luu (kadu\|vähenemine\|piisavus) | implantaatidega ravi edukuse võti on lõualuu hulk ja kvaliteet implantatsiooni piirkonnas | | N |
| lõualuu | lõualuu plastika parandab oluliselt ravi esteetilist ja funktsionaalset tulemust | | N |
| maarjamõisa\|maarjamõisas\|erakorraline\|hädaabi\|väljakutse\|öösel\|öösiti\|kellaaeg ... hiline | maarjamõisas on võimalik ööpäevringselt valu korral abi saada (telefon 112) | | Y |
| maksumus\|maksma\|kulukas\|kirves\|pappi\|raha\|tasu\|hind\|hinnakiri\|kallis\|eur\|euro\|summa | täpse hinna saab arst öelda peale esmast kontrolli, hindadest saate ülevaate siit [ava hinnakiri] | | N |
| (murduma\|murdma\|murd\|murdunud\|pooleks) ... hammas | murtud hammast saab üldjuhul parandada, peaksite seda arstile näitama | | Y |
| mäda\|mädapunn\|vill\|haavand\|mädanema\|punn\|stomatiit | tegemist võib olla ka suu limaskesta põletikuga, oleks mõistlik seda arstile näidata | STOMATIIT | N |
| mädanik\|põletik\|(ige ... lahti)\|tasku\|tundlik\|suur | soovitame tulla visiidile, vaatame teie olukorra üle | | N |
| vahe\|valuline\|turses\|pakitseb\|esihammas\|purihammas\|silmahammas | | | |
| nakkav\|nakkama\|pere\|perekond | nakkuse kohta oskab täpsemalt öelda arst, mikroobid ikka levivad pereliikmete vahel | | N |
| nädalavahetus\|laupäev\|pühapäev | nädalavahetusel saab erandjuhul vastuvõtule tulla, kui arstiga eelnevalt kokku leppida | NÄDALAVAHETUS | N |
| närimistubakas | närimistubakas on paikselt kasvajatekkimise riskitegur | | N |
| olema ... valus | tehakse tuimestus ja ei ole valus | | N |
| ortodontia\|hambumus\|hambumusprobleem | ortodontiaga (sh breketitega) meie ei tegele, vaadake näiteks [www.****.ee] | | N |
| paistes\|paistetus\|paise\|paiste | paistes? paistetus võib olla päris tõsine olukord, tuleks tulla arsti vastuvõtule | | N |
| panoraam.*\|röntgen | röntgeni saate teha **** Hambakliinikus | | N |
| paradontax\|parodontaks\|paradontaks\|parodontax\|paratontax\|parotontax\|paratontaks\|parotontaks | parodontax on üks paljudest looduslikest hambapastadest, kasutatakse profülaktikas | | N |
| parodontoloog.* | julgen soovitada parodontoloogi - Dr. **** (asub Tartus ****) | | N |

| Pattern | Response | State | X |
|---|---|---|---|
| protees\|proteesija\|jäljend\|proteesimine\|proteesima\|proteesid\|kroon\|portselan\|kroonimine\|hambaprotees | proteesidega (kroonidega) variandid on kirjas siin vt. [loe proteesimisest] | PROTEESID | N |
| sektor | suu jaotatakse hambaarstide keelepruugis neljaks sektoriks (ülemine parem- ja vasakpoolne ning alumine parem- ja vasakpoolne) | | N |
| sild | sildprotees tsementeeritakse tugihammastele | PROTEESID | N |
| soodapesu\|sooda\|tume plekk\|pigment | soodapesuga saab arst eemaldada pigmente ja hambakivi | | N |
| soovitama | soovitame tulla visiidile, vaatame teie olukorra üle | | N |
| ... | | | |
| ... | | | |
| ... | | | |

**The states of Zelda** are shown in the following table. The column "Order" shows the position of the response in the stack. So, if the state = "IGEMERAVI" the first sentence to be presented to the user would be "igemeravi sisaldab erinevaid protseduure".

| State | Response | Order |
|---|---|---|
| HAMBAKIVI | kui jätta hambakivi eemaldamata, siis on oht igemehaiguste tekkeks | 1 |
| HAMBAKIVI | hambakivi sisaldab suurel hulgal baktereid, mille tagajärjeks on veritsevad ning valutavad igemed, paha maitse suus ning halb hingeõhk | 2 |
| HAMBAKIVI | ise koduste vahenditega hambakivi üldjuhul eemaldada ei õnnestu | 3 |
| HAMBAKIVI | hambakivi eemaldamisel kasutatakse erinevaid meetodeid | 4 |
| IGEMERAVI | igemeravi sisaldab erinevaid protseduure | 1 |
| IGEMERAVI | alustuseks uuritakse kui ulatuslik on teie põletik ning tehakse raviplaan | 2 |
| IGEMERAVI | enne esmast visiiti tuleks teha panoraamröntgen | 4 |
| IGEMERAVI | röntgeni saate teha **** Hambakliinikus | 5 |
| IMPLANTAAT | üks hetk, kohe räägin teile lühidalt implantaadi paigaldusest ... | 1 |
| IMPLANTAAT | kõigepealt asetatakse implantaat, mis sarnaneb kruviga, teie lõualuusse | 2 |
| IMPLANTAAT | see kaetakse ajutise hambaga, mis varjab tühimiku | 3 |
| IMPLANTAAT | implantaadil lastakse 3-6 kuud luustuda, misjärel kinnitatakse selle külge hambakroon | 4 |
| IMPLANTAAT | hinnad on ligikaudu sellised, üks tehisjuur **** eur, selle paigaldus **** eur, kroonimine **** eur, kroon **** eur | 5 |
| JÄRELMAKS | järelmaks toimib nõnda, et töö jagatakse mitmesse osasse ning tasute järk-järgult | 1 |
| JÄRJEKORD | kui täna helistada, siis arvan, et saate aja järgmiseks nädalaks | 1 |
| JÄRJEKORD | enne esmast visiiti tuleks teha panoraamröntgen | 2 |
| JÄRJEKORD | röntgeni saate teha **** Hambakliinikus | 3 |
| KODUNE HOOLDUS | kodune hooldus seisneb hambavahede puhastamises niidiga | 1 |
| KODUNE HOOLDUS | hambahaiguste ärahoidmiseks on tähtis, et hammaste igalt pinnalt eemaldataks katt hoolsalt ja korrapäraselt | 2 |
| KODUNE HOOLDUS | ei tohi vigastada iget teravate tikkudega ja ei tohi kasutada hambaharja, mis on kõvade harjastega | 3 |
| LAMINAAT | laminaatide abil on võimalik esihammaste kuju ja värvust parandada | 1 |
| NARKOOS | kahjuks ei oska soovitada kliinikut, kus pakutaks hambaravi üldnarkoosis | 1 |
| NÄDALAVAHETUS | üldjuhul nädalavahetusel vastuvõttu ei toimu | 1 |

# Appendix D

## Sample conversations with the ADS based systems

This appendix contains several sample conversations with the systems that were built on the ADS framework – Zelda and Alfred.

**Conversation no 1.** This is a conversation with Alfred. The system was assisted by the live operator. The last column contains a "X" when the response originated from the live operator.

| | | | |
|---|---|---|---|
| 13:33.43 | Alfred | tere | |
| | | *hello* | |
| 13:33.53 | Alfred | hetkel on Tartu kinokavades kokku 21 erinevat filmi | |
| | | *there are 21 different movies in Tartu currently* | |
| 13:34.09 | Alfred | kui seda näinud ei ole, siis ehk tasuks minna: täna, kell 14:05 – "Viimane laul", kinos Cinamon | |
| | | *you might be interested in this one: today at 14:05 "Last Song" in Cinamon* | |
| 13:34.52 | User | mis veel täna on? | |
| | | *what else is played today?* | |
| 13:35.03 | Alfred | selge, et siis täna? – täna, kell 14:15 – "Punane elavhõbe", kinos Cinamon | |
| | | *ok, today – at 14:15 "Red Quicksilver" in Cinamon* | |
| 13:35.23 | User | natuke hiljem | |
| | | *a little later* | |
| 13:35.28 | User | näiteks kell 17 | |
| | | *let's say at 17* | |
| 13:35.37 | Alfred | et sulle sobiks siis kell 17? – täna, kell 17:05 – "Lihtsalt liiga kuum", kinos Cinamon | |
| | | *so, you would like at 17? – today at 17:05 "She's Out of My League" in Cinamon* | |
| 13:36.09 | User | on see komöödia? | |
| | | *is it a comedy?* | |
| 13:36.17 | Alfred | jah | X |
| | | *yes* | |
| 13:37.36 | Alfred | sõber mainis, et seda vaadates jäi ta magama: täna, kell 19 – "Pärsia Prints: Ajaliiv", kinos Cinamon | |
| | | *a friend told me that he fell asleep while watching this one: "Prince of Persia: The Sands of Time"* | |
| 13:38.09 | Alfred | ok, ootan siis vaikselt, söö rahulikult ... | |
| | | *ok, I'll wait quietly, go ahead and have your meal ...* | |

**Conversation no 2.** This is a conversation with Zelda (virtual dental consultant). The system was not assisted by the live operator, as assistance was not necessary. During the conversation one spelling error was corrected by the system (implanaati ⇒ implantaati).

| | | |
|---|---|---|
| 16:11.45 | Zelda | tere! minult saate küsida hambaravi, igemeravi, implantaatide, proteeside kohta ...<br>*hello, you can ask me about dental care, gum disease treatment, dental implants and prosthetics* |
| 16:11.52 | Zelda | igaks juhuks ütlen, et olen arvuti (juturobot), mitte inimene<br>*just to warn you – I'm a machine (chatbot), not a human,* |
| 16:12.08 | Zelda | kui vastamisel hätta jään, siis võite oma küsimused ka arstidele jätta: [saada küsimus siit]<br>*in case I fail to respond, you can leave your questions to the doctor by [sending a message]* |
| 16:12.15 | Zelda | minu nõuanded ei asenda hambaarsti visiiti, vaid on mõeldud patsientidele eelkõige vajaliku eelinfo kogumiseks ja täiendavate teadmiste saamiseks<br>*my advice is not to be considered as a replacement of a real dental consultation, but as an alternative way to get information* |
| 16:12.41 | User | mul on alalõuas eemaldatud hammas, juba 20 aastat tagasi<br>*I have a tooth removed from the lower jaw 20 years ago* |
| 16:12.57 | User | kas on võimalik sinna **implanaati** panna<br>*is it possible to get a dental implant* |
| 16:13.00 | Zelda | põhjalikum info on siin: [loe implantaatidest]<br>*more information can be found here [read about dental implants]* |
| 16:13.10 | Zelda | üks hetk, kohe räägin teile lühidalt implantaadi paigaldusest ...<br>*just a moment, I'll tell you shortly about the installation of dental implants* |
| 16:13.25 | Zelda | kõigepealt asetatakse implantaat, mis sarnaneb kruviga, teie lõualuusse<br>*an implant, which is similar to the screw, will be placed into your jaw* |
| 16:13.32 | Zelda | see kaetakse ajutise hambaga, mis varjab tühimiku<br>*this is covered with a temporary tooth, which conceals the gap* |
| 16:13.36 | Zelda | implantaadil lastakse 3–6 kuud luustuda, misjärel kinnitatakse selle külge hambakroon<br>*the implant needs about 3–6 months to ossify, and then a crown is attached* |
| 16:13.54 | Zelda | hinnad on ligikaudu sellised, üks tehisjuur **** eek, selle paigaldus **** eek, kroonimine **** eek, kroon **** eek<br>*prices are: implant **** eek, installation **** eek, crowning **** eek, crown **** eek* |
| 16:13.58 | User | tuleksin arsti juurde konsultatsioonile aga kuhu?<br>*I would like to see a dentist for a consultation, but where should I go?* |
| 16:14.00 | Zelda | Dr. **** number on ****<br>*the phone number of Dr. **** is ***** |
| 16:14.13 | Zelda | meie asukoha leiate siit [vaata kontaktandmeid]<br>*our dental clinic is located here [see the location]* |
| 16:14.21 | Zelda | enne esmast visiiti tuleks teha panoraamröntgen<br>*you should get a panorama x-ray taken prior to the first visit* |
| 16:14.22 | User | kas ta on Eesti parim :)<br>*is this doctor the best in Estonia :)* |
| 16:14.38 | Zelda | röntgeni saate teha **** Hambakliinikus |

| | | |
|---|---|---|
| | | *the panorama x-ray can be taken in **** Clinic* |
| 16:14.44 | Zelda | visiiditasu on esmasel konsultatsioonil **** kuni **** eek (siin sisaldub ka raviplaani koostamine)<br>*the fee for the initial consultation is **** – **** eek (this includes the preparation of the treatment plan)* |
| 16:15.23 | User | aga Tallinnas?<br>*what about someone in Tallinn?* |
| 16:15.31 | Zelda | ei julge kedagi Tallinnas soovitada, teadmata teie probleemi täpseid asjaolusid<br>*I'm not in the position to recommend anyone in Tallinn without knowing the detailed circumstances of your case* |
| 16:15.44 | User | ok |
| 16:15.49 | Zelda | ok |

# Appendix E

## Glossary of terms used in the thesis

The definitions of these basic terms are taken from the Natural Language Processing Dictionary [Wilson, 2000] and from the International Dictionary of Artificial Intelligence [Raynor, 1999]. The definitions are adjusted by the author of the thesis and explained in the context of this thesis.

**ambiguity (word-sense ambiguity)** – An ambiguity is a situation where more than one meaning is possible in a sentence. Word-sense ambiguity is a kind of ambiguity where what is in doubt is what sense of a word is intended. In this thesis, the problem of ambiguity arises in spell-checking and temporal resolution.

**base form** – A base form or a root is the primary lexical unit of a word (or a word family), which carries the most significant aspects of semantic content and cannot be reduced into smaller constituents. In this thesis, words are reduced to their base forms to simplify the process of semantic resolution.

**human-assisted chat** – A chat where the dialogue system allows a human agent to help a number of simultaneous chat sessions by having an AI-engine (the module that aims to implement the artificial intelligence abilities) handle the bulk of common, repeat questions. This approach is implemented in the ADS framework.

**knowledge base** – A knowledge base is a special kind of database for knowledge management, providing the means for the computerized collection, organization, and retrieval of knowledge. In this thesis, the knowledge base is a collection of facts and information that define the domain.

**knowledge engineering** – Knowledge engineering is an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise. In this thesis, it is used to refers to the building, maintaining and development of the knowledge-base.

**knowledge representation** – Knowledge representation and reasoning is an area of artificial intelligence whose fundamental goal is to represent knowledge in a manner that facilitates inferencing (i.e. drawing conclusions) from knowledge. In this thesis, the knowledge about a domain is represented in two ways: by declarative rules and by procedural program code.

**morphology** – Morphology is the identification, analysis and description of the structure of morphemes and other units of meaning in a language like words, affixes, and parts of speech. In this thesis, the morphological analyzer [Kaalep and Vaino, 2001] is used to reduce the words to their base forms.

**n-gram** – An n-gram is a subsequence of n items from a given sequence. The items in this thesis are words or base forms of the words. In this thesis, ngrams are used for

generating permutations of words to handle the word order problem. N-grams can be also used in probabilistic models.

**pattern matching** – Pattern matching is the act of checking some sequence of tokens for the presence of the constituents of some pattern. In this thesis, the user input is checked for the presence of certain words or phrases. If a certain pattern is recognized in the user input then a certain action is performed.

**regular expression** – Regular expressions provide a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor, a program that either serves as a parser generator or examines text and identifies parts that match the provided specification. In this thesis, regular expressions are used to define patterns. If a certain pattern is recognized in the user input then a certain action is performed.

**semantic resolution** – Semantic Resolution is focused on the processing of linguistic meaning. In this thesis, the semantic resolution is meant to describe the process of capturing the meaning from the user input.

**stemming** – Stemming is the process of reducing the word to its base form (also known as root or stem), by removing its suffixes. In this thesis, the morphological analyzer [Kaalep and Vaino, 2001] is used to reduce the words to their base forms. The base forms are used in the patterns, thus keeping the patterns short and simple.

**stop word** – Stop words are words which are filtered out prior to, or after, processing of natural language data (text). There is not one definite list of stop words which all tools use, if even used. The ADS Framework includes an option to define stop words. The removal of stop words can improve the phrase search.

**temporal expression** – A temporal expression in a text is a sequence of tokens (words, numbers and characters) that denote time, that is express a point in time, a duration or a frequency. In this thesis, the temporal expressions are discussed in the context of temporal resolution – the process of defing which point of time was meant by a user input.

# Curriculum Vitae

## Margus Treumuth


Born: February 5, 1976, Tartu
Citizenship: Estonian
Marital Status: married, 1 son and 1 daughter
Contacts: (+372) 52 00883, treumuth@ut.ee


### Education

| | |
|---|---|
| 1983–1993 | Võru Kreutzwald Gymnasium |
| 1993–1994 | Wake Forest-Rolesville High School (NC, USA), secondary education |
| 1994–1995 | University of Tartu |
| 1995–1996 | Campbell University (NC, USA) |
| 1996–2002 | University of Tartu, BSc, Computer Science |
| 2002–2004 | University of Tartu, MSc, Computer Science |
| start. 2004 | University of Tartu, PhD student, Computer Science |

### Working Experience

| | |
|---|---|
| 2000–2006 | Aprote Ltd./WM-Data Ltd., programmer |
| start. 2006 | sole proprietor (Margus Treumuth FIE), programmer |
| start. 2007 | Estonian Unemployment Insurance Fund, IT manager |

# Elulookirjeldus

## Margus Treumuth

| | |
|---|---|
| Sünniaeg ja -koht: | 05.02.1976, Tartu |
| Kodakondsus: | Eesti |
| Perekonnaseis: | abielus, 1 poeg ja 1 tütar |
| Kontakt: | (+372) 52 00883, treumuth@ut.ee |

### Haridus

| | |
|---|---|
| 1983–1993 | Võru Kreutzwaldi Gümnaasium |
| 1993–1994 | Wake Forest-Rolesville High School (NC, USA), keskharidus |
| 1994–1995 | Tartu Ülikool |
| 1995–1996 | Campbell University (NC, USA) |
| 1996–2002 | Tartu Ülikool, BSc, informaatika |
| 2002–2004 | Tartu Ülikool, MSc, informaatika |
| alates 2004 | Tartu Ülikool, doktorant, informaatika |

### Töökogemus

| | |
|---|---|
| 2000–2006 | AS Aprote/AS WM-Data, programmeerija |
| alates 2006 | Margus Treumuth FIE, programmeerija |
| alates 2007 | Eesti Töötukassa, infosüsteemi arendusjuht |

# DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. **Mati Heinloo.** The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu, 1991, 23 p.
2. **Boris Komrakov.** Primitive actions and the Sophus Lie problem. Tartu, 1991, 14 p.
3. **Jaak Heinloo.** Phenomenological (continuum) theory of turbulence. Tartu, 1992, 47 p.
4. **Ants Tauts.** Infinite formulae in intuitionistic logic of higher order. Tartu, 1992, 15 p.
5. **Tarmo Soomere.** Kinetic theory of Rossby waves. Tartu, 1992, 32 p.
6. **Jüri Majak.** Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu, 1992, 32 p.
7. **Ants Aasma.** Matrix transformations of summability and absolute summability fields of matrix methods. Tartu, 1993, 32 p.
8. **Helle Hein.** Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu, 1993, 28 p.
9. **Toomas Kiho.** Study of optimality of iterated Lavrentiev method and its generalizations. Tartu, 1994, 23 p.
10. **Arne Kokk.** Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu, 1995, 165 p.
11. **Toomas Lepikult.** Automated calculation of dynamically loaded rigid-plastic structures. Tartu, 1995, 93 p, (in Russian).
12. **Sander Hannus.** Parametrical optimization of the plastic cylindrical shells by taking into account geometrical and physical nonlinearities. Tartu, 1995, 74 p, (in Russian).
13. **Sergei Tupailo.** Hilbert's epsilon-symbol in predicative subsystems of analysis. Tartu, 1996, 134 p.
14. **Enno Saks.** Analysis and optimization of elastic-plastic shafts in torsion. Tartu, 1996, 96 p.
15. **Valdis Laan.** Pullbacks and flatness properties of acts. Tartu, 1999, 90 p.
16. **Märt Põldvere.** Subspaces of Banach spaces having Phelps' uniqueness property. Tartu, 1999, 74 p.
17. **Jelena Ausekle.** Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu, 1999, 72 p.
18. **Krista Fischer.** Structural mean models for analyzing the effect of compliance in clinical trials. Tartu, 1999, 124 p.

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. **Eno Tõnisson.** Solving of expession manipulation exercises in computer algebra systems. Tartu, 2002, 92 p.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.

42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.
43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.

63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
65. **Larissa Roots.** Free vibrations of stepped cylindrical shells containing cracks. Tartu 2010, 94 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
68. **Olga Liivapuu.** Graded q-differential algebras and algebraic models in noncommutative geometry. Tartu 2011, 112 p.
69. **Aleksei Lissitsin.** Convex approximation properties of Banach spaces. Tartu 2011, 107 p.
70. **Lauri Tart.** Morita equivalence of partially ordered semigroups. Tartu 2011, 101 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.