Iterative Development of an Information-Providing Dialogue System

by

Pontus Johansson

Department of Computer and Information Science

Linköping University

SE-581 83 Linköping, Seden

Linköping 2001

Abstract

A functional prototype of a dialogue system with a natural language interface was implemented for the Nokia Mediaterminal. The prototype was developed from a generic dialogue system framework called MALIN. The purpose of this rapid prototyping approach is to establish what kind and amount of work that needs to be carried out in order to customize a dialogue system based on the MALIN framework for the chosen domain. The method used is developed for dialogue system construction, and utilizes an iterative approach where conceptual design and implementation are viewed as two distinct steps carried out in parallel. The empirical basis for the design and implementation is a dialogue collection and a corpus provided by the domain knowledge base content. Domain knowledge base content is detailed information about one week of television programmes represented in a relational database.

The prototype is confined to allow for single utterances dealing with simple task requests. That is, dialogue management is allowed for designwise, but is not implemented, in order to incorporate it in subsequent iterations. The design and implementation resulted in a modularized system, in which the individual components can be exchanged and/or modified without having to rewrite the whole system.

The experiences of the design and implementation work results in suggestions for future research, and a suggestion for a new system architecture that allows for increased portability, scalability, and

transportability. These aspects are accounted for from two perspectives: the linguistic aspect and the domain aspect.

Acknowledgements

I would like to thank my advisors Lena Strömbäck and Arne Jönsson for their support. They have guided me, answered questions, and provided me with useful comments on my work.

Lars Degerstedt has patiently discussed Java coding in general and the MALIN system in particular, which has been very helpful for my work. Håkan Johansson has also been a great help with Lisp and the chart parser.

Magnus Lindwall, Magnus Henriksson, and Roberto Barriga have helped me with the techniqualities of setting up the MySQL database, for which I am grateful.

I would also like to thank Martin Wiman for testing the first version of the grammar rules, thus helping me extend the project corpus, and for acting as my opponent in the defence of this thesis.

Finally, I would also like to thank Cissi for her support, patience, and useful comments on the draft.

Linköping, 2001-07-03

Pontus Johansson

Table of Contents

Figures and Tables Index

1 Introduction

"The only source of knowledge is experience."

-- Albert Einstein

1.1 BACKGROUND

As the amount of information technology in our casual home environment increases, there is a growing demand for efficient interaction and ease-of-use. Many of today's households have an Internet connection and various entertainment technologies that utilize it. It is expected that various machines and information systems in the "e-home" environment will be further integrated in the near future.

This thesis is a part of the MIINA (Multimodal Interaction for Information Appliances) project. MIINA is a joint project between the Natural Language Processing group at Department of Computer and Information Science, Linköping University, and Nokia Home Communications. The project aims to investigate and develop multimodal interaction systems for the future "e-home". Specifically, set of tasks that relates to the management of information for the household is to be investigated. The long-term vision is to integrate the systems within a common e-home framework. The approach taken in the MIINA project is that a fruitful research strategy will start

from specific examples, and work towards a common framework, instead of trying to develop such a framework in a top-down fashion. A specific example is the integration of Internet and digital television, which is the topic of this thesis.

More information about the MIINA project and its goals is available at: `http://www.ida.liu.se/~nlplab/miina/`.

1.1.1 Nokia Home Communications

Nokia Home Communications (NHC) is part of Nokia Ventures Organization, employs 560 people and is headquartered in Sweden. NHC works with information technology and entertainment – coined "infotainment" – for households. Main product development sites are located in Sweden, Finland and the US, with a well-established worldwide sales organization. The staff in Linköping is made up of 300 people from some 20 countries. These employees, who are specialists in a broad range of fields, form networks through which they work in close cooperation with colleagues around the world. The goal of NHC is to establish an open standard for home entertainment based on open Internet technologies.

Recently NHC released the Nokia Mediaterminal, a system that integrates Internet and digital television into one module. This technology makes it possible to seamlessly surf the Internet, watch TV, read and send e-mail, play music, and chat by using a remote control. The end user of the Mediaterminal is a private person in a household who uses the Mediaterminal on a casual basis. Since the target group for the Mediaterminal thus is very large and versatile, and the amount of information available to the user is vast, there is an increasing demand of usability on the interface for television programme queries. Usability and ease-of-use are design aspects stressed by Nokia in general, and NHC in

particular. It is desirable to develop a system that allows a wide range of users to extract relevant, correct and manageable information efficiently – without them having to learn a formal query language. To come to terms with this a natural language interface to a database, containing information about television programmes, has been suggested. The vision is that the user should be able to ask questions about television programmes, channels, categories, and starting times in natural language; and get tailored, relevant, correct and manageable information back from the system.

1.1.2 Natural Language Processing Laboratory

At the University of Linköping there is a research group for natural language processing working at the Natural Language Processing Laboratory (NLPLAB). NLPLAB is affiliated with the Department of Computer and Information Science, Division for Human-Centered Systems that works on natural language processing and related areas of Computational Linguistics and Cognitive Science. The group was formed in 1986. The research conducted within NLPLAB is spanning both theoretical and applied areas of natural language processing. One of the main areas is focused on the development of dialogue systems.

NLPLAB has developed a range of software systems and modules. Among them is a unification-based chart parser that is used to analyze text. There also exists a rule-based dialogue system framework, which manages dialogues between a user and background system.

More information can be found at the NLPLAB Web site at: `http://www.ida.liu.se/~nlplab/`.

1.2 PURPOSE

The problem space for this thesis is typed human-computer interaction with a dialogue system with a natural language interface. The domain is television programmes and tableaus for a Mediaterminal/digital television box.

The aim of this thesis is two fold. First, it is to explore how much and what kind of work that needs to be done when customizing a generic dialogue system framework to a new natural language, a new domain, and a new computational platform. A second aim is to implement a functional dialogue system prototype with a natural language interface to demonstrate the use of this interaction technique for the Nokia Mediaterminal.

A functional prototype is to be designed and implemented, and the experiences drawn from the development provide the answers to the questions posed in this thesis. The research questions are further developed in chapter 3 Problem Statement.

1.3 OVERVIEW

This thesis report is outlined as follows:

Chapter 2 Related Research gives an account of the theoretical framework relevant for this thesis. It deals with dialogue systems, feature structures, grammar formalisms and parsing, knowledge representation and natural language generation.

Chapter 3 Problem Statement develops the purpose stated above into two main aspects and poses the questions this thesis aims to answer.

Chapter 4 The MALIN Framework gives a description of the generic dialogue system framework that has been developed at the NLPLAB at the University of Linköping.

Chapter 5 The TV-Programme Dialogue System gives a detailed account for the procedure of building the prototype. It covers the corpus work, the design stage and the implementation stage. The chapter is concluded with a clarifying example run of the system.

Chapter 6 Results describes the results of the system development.

Chapter 7 Discussion discusses the results and provides the answers to the questions stated in chapter 3 Problem Statement. Suggestions of how to improve the extensibility and functionality are given.

In **chapter 8 Conclusion**, a summary of this thesis contributions is made, as well as guidelines for future research. This chapter also includes a suggestion of a new system architecture that can be employed for the next version of the dialogue system prototype.

A glossary consisting of terms, concepts, and abbrevations is found in **Appendix A** for easy reference.

2 Related Research

This chapter gives an account of the state of the art of the theoretical research carried out in the fields relevant to this thesis. Section 2.1 Preliminaries defines a few key concepts and terms. In section 2.2 Dialogue Systems a brief introduction to dialogue systems is presented. Two important concepts for parsing and knowledge reasoning are then described. Section 2.3 Feature Structures and Unification deals with feature structures and unification, while section 2.4 Grammar Formalisms and Parsing describes the grammar formalisms that form the basis for chart parsing, which is also described. Then an overview of some terms in the field of representing and retrieving various kinds of knowledge follows in section 2.5 Representing and Retrieving Knowledge. Finally, natural language generation is described in section 2.6 Natural Language Generation.

2.1 PRELIMINARIES

In order to make the rest of this thesis coherent, a few key concepts need to be explained, and clarified:

This thesis deals with aspects of Language Technology, and not Speech Technology. *Speech technology* refers to language processing on the level of Phonology and Phonetics and is founded in Acoustics, Electrical Engineering and Computer Science. *Language technology* is the process of analyzing textual input in terms of syntax, morphology, and semantics.

The foundations for this lie in Computational Linguistics, Psychology and Computer Science. The systems described herein are dialogue systems that are limited to the processing of written – or typed – interaction (Jurafsky & Martin, 2000).

An *utterance* is a generic term referring to any mode of communication. That is, an utterance may be typed or signed for example (Russel & Norvig, 1995).

A *dialogue* is a joint activity between two entities (e.g. a user and a computer system). What makes dialogues different from other types of discourse is that they are concerned with grounding and turn-taking. *Grounding* is to establish a base of mutual understanding, or common ground, well enough for the current purposes. In the case of dialogues, this often attributes to confirmation of understanding of a conversational counterpart's utterance. *Turn-taking* in a dialogue can be strict (no interruption is allowed) or flexible. For the purpose of this thesis, the interaction is typed on a keyboard, and the system described herein enforces strict turn-taking (i.e. interruptions and parallel utterances are not allowed). A dialogue system allowing spoken interaction requires more advanced concepts (e.g. incremental speech recognition and parsing for flexible turn-taking) (Clark, 1996).

Natural language refers to human languages that can be written and/or spoken (e.g. English, Swedish and Latin), whereas *formal languages* are rigidly defined and artificial (e.g. Logic, Java, SQL and Lisp). (Jurafsky & Martin).

Domain is an area of knowledge, for which a specific system is developed. A *genre* is a type of discourse with certain specific features and content. *Sub-language* is the set of a natural language that is used within a genre (Jurafsky & Martin).

2.2 DIALOGUE SYSTEMS

The general definition of a *dialogue system* (sometimes called a conversational system) is, according to Allen et. al (1998), a computer system with a natural language interface that allows the user to communicate in his or her natural language, and involves more or less complex dialogues.

The range of complexity in dialogue systems is wide. Dialogue system complexity can be classified as shown in Table 1:

Table 1 Classification of dialogue complexity range. From Allen et al (2001b).

Technique Used	Example Task	Task Complexity	Dialogue Phenomena handled
Finite-state Script	Long-distance dialing	least complex	User answers questions
Frame-based	Getting train timetable information		User asks questions, simple clarifications by system
Sets of Contexts	Travel booking agent		Shifts between predetermined topics
Plan-based Models	Kitchen design consultant		Dynamically generated topic structures, collaborative negotiation sub-dialogues
Agent-based Models	Disaster relief management	most complex	Different modalities (e.g. planned world and actual world)

To put things in context: most dialogue systems constructed to date dwell in the first two regions. Furthermore, the gap between theoretical (i.e. non-implemented) dialogue systems and existing implemented dialogue

systems is vast. There are several capabilities that dialogue systems of various complexities should feature according to various literature sources: interpretation, handling tasks, requests, initiative turn-taking, focus and dialogue history handling, domain knowledge reasoning, response generation etc. There are to the author's knowledge very few implemented dialogue systems to date that feature all the capabilities above. It can therefore seem that dialogue systems of little complexity are hardly worth the name dialogue systems, at least not in the sense that the word "dialogue" is normally used. Also, since there is a separation between language technology and speech technology, the term dialogue system is sometimes denoting spoken, and sometimes written, and sometimes multimodal, interaction. Henceforth, a dialogue system refers to a computer system that interacts with users through natural language interpretation and generation, where the user is not restrained to use pre-defined commands. The modality may be spoken, written, typed or even signed, but for the purpose of this thesis only typed interaction is discussed.

There are several other classifications available on dialogue systems and their capabilities, but for the purpose of this thesis the classification in Table 1 should be sufficient.

2.2.1 NATURAL LANGUAGE INTERFACES

Jacobs and Rau (1988) claim that natural language interfaces are "making the task of communicating with the information source easier, allowing a system to respond to a range of inputs, possibly from inexperienced users, and to produce more customized output." (p. 85).

A natural language interface places no constraints on the user, since he or she can provide the system with any string of words. A system with no constraints requires error handling (Norman, 1988). Allen et al (1998)

suggest that a dialogue system that does not constrain the user has as a goal to expand the user's option in the interaction. Watt (1968, in Jönsson, 1993) defines a system as *habitable* if the user never feels that he or she is restrained by prohibited sentences in the interaction. A delicate problem when building a dialogue system is to design a habitable system with limited coverage even so (see section 2.2.2 Linguistic Coverage).

Recently, natural language interfaces for relational databases have emerged. According to Androutsopoulos et al (1994), the main advantages with natural language interfaces for database information retrieval are:

- Natural language is not an artificial language, and requires no training from a native speaker. The user does not have to spend time learning to use a new interaction technique.
- Natural language is a better medium for handling anaphora and elliptical expressions, and is suitable for dialogue systems.
- Natural language is better for questions dealing with negations and universal quantifications.

Experiments show that natural language interfaces for database information retrieval seem to be better in queries where data from many tables need to be combined, and in queries that are not similar to the ones that users encounter during training sessions (Bell & Rowe, 1992). A conclusion drawn from this is that the use of natural language interfaces is an effective method of interaction for casual users, who have no wish or ability to learn an artificial querying language, such as SQL. Natural language interfaces are suitable for users performing question-and-answer tasks, in a restricted domain (Androutsopoulos et al, 1994).

2.2.2 LINGUISTIC COVERAGE

As mentioned in section 2.2.1 Natural Language Interfaces, it is desirable that a dialogue system with a natural language interface is habitable. Even though Jönsson (1993) approves of, and recommend habitable systems, he stresses the importance of not having the system mimicking general human conversation. The main reason for this is that it is too slow, and that it can provide the user with an erroneous model of the system's capabilities (Jönsson, 1997). Androutsopoulos et al (1994) agree and claim that the user could be misled by the system's ability to process language, and assume that the system possesses intelligence. In contrast, the work of Allen et al (1998), aims to build dialogue systems that mimic human conversation, but in a specialized domain.

It is not feasible to regard a dialogue system as a general-purpose system. The reasons for this are two: First, the linguistic component of a system that would allow unrestricted natural language dialogues would be massive. It would require a very tedious configuration phase, and require a lot of computational power. Second, the knowledge base for a system with an unrestricted domain would require even more work and power, and chances are that the bulk of such a system would not be used (Jönsson, 1993). Each specific application has its own requirements. Identifying the system's domain, end users, and type of task(s) are therefore essential. Allen et al (1998; 2001b) bring up the notion of *practical dialogues*. A practical dialogue is focused on accomplishing a concrete and well-defined task, such as task-oriented dialogues, information-seeking dialogues and command/control dialogues. The distinction between full natural language, and the sub-language of a practical dialogue is important to make, since the effort and system requirements required to model them differ. This is concluded in *The Practical Dialogue Hypothesis* (Allen et. al, 1998): "The conversational

competence required for practical dialogues, while still complex, is significantly simpler to achieve than general human conversational competence." (p. 2).

Depending on the complexity of the task, the syntactic and semantic approach can be more or less detailed. The system should handle the level of complexity of the sub-language associated with the task.

The sub-language of a specific task is described in a project *corpus*. The corpus is a set of texts that are representative in respect to genre and domain, and that have been gathered for the purpose of system development. According to Kelley (1983, in Jönsson, 1993) only those phenomena actually occurring in the corpus need to be considered when deciding on linguistic coverage. The catch to this is that the corpus needs to be rather large in order to account for every action taken by every potential user.

According to *The Principle of Compositionality*[1], the meaning of a sentence is the summary of the meanings of its parts. Not only does the meaning of the individual words contribute; the ordering, grouping and relations between them also bear meaning for the sentence as a whole. To illustrate the need for making allowance for word order, consider the following example: The sentences "I eat what I see" and "I see what I eat" contain the same words, but their meanings are different due to the ordering of the words. Ignoring the word ordering, and treat them as identical "bags of words", would make them mean the same thing, (Jurafsky & Martin, 2000).

[1] According to Jurafsky and Martin (2000) the principle of compositionality is normally referred to Frege, but that there appears to be little reason for this ascription. Janssen (1997, in Jurafsky & Martin) discusses this topic in more detail.

2.2.3 DOMAIN COVERAGE

Deciding how to analyze the syntax and semantics of natural language depends on the complexity of the task at hand. Simple information extraction tasks may only require identification of a few keywords, while more advanced tasks require full syntax and extensive semantics. Knowledge engineering is present in all systems where domain-specific issues exist. Cowie and Lehnert (1996) stresses that the knowledge coverage decision is very important, since this can help avoiding "knowledge engineering bottlenecks". As mentioned in the previous section, a dialogue system with a very large – or unrestricted – knowledge base requires extensive knowledge engineering. This takes time and effort to perform. There is a power in what Cowie and Lehnert call "shallow knowledge", which is a term developed to describe a methodological approach to remove knowledge engineering bottlenecks, and thus minimizing the required knowledge engineering. To minimize a system's knowledge engineering, Cowie and Lehnert refer to minimizing the effort associated with knowledge acquisition. To use shallow knowledge in information extraction systems means that ad hoc-rules are specified, and bound to the domain. This approach has been criticized, since it implies loss of generality. Still, Cowie and Lehnert justify it since "shallow knowledge is so cheap to acquire that it is cost-effective to treat it as a disposable artifact […] and we need not worry about knowledge engineering bottlenecks." (p. 86). In comparison with other pieces of overhead associated with constructing computer systems (such as code optimization, documentation and user interface design) the cost of the shallow knowledge approach becomes rather small.

2.2.4 System Architecure and Modularization

Researchers generally agree upon the need to modularize dialogue systems (Allen et al; 1998; 2001a; 2001b; Degerstedt & Jönsson, 2001; McRoy et al, 1999; Flycht-Eriksson & Jönsson, 2000, among others). The exact layout of the modules varies depending on the system. A module has a specific responsibility (such as interpretation, response generation, or knowledge management) and interacts with the other modules according to native rules, or rules in the controlling module. Various approaches to the controlling module have been suggested. A separate module, called a hub or facilitator for controlling the information flow can be used (cf. Flycht-Eriksson, 2000), or the controlling function can be put in the dialogue management module (Jönsson, 1993). Even though the exact module design differs, most dialogue systems make a distinction between a linguistic system and a background – or knowledge – system. Figure 1 shows a simple and generic dialogue system architecture.

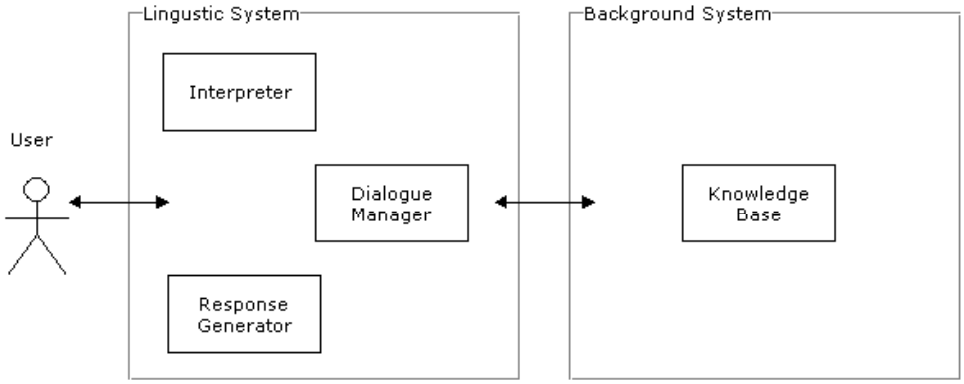

Figure 1 Generic dialogue system architecture.

It proves advantageous to further divide the linguistic module into sub-units. Typically, there is an interpreter (parser) with a domain-dependent lexicon, a response generator and a dialogue management module included in the linguistic component. According to Thompson and

Thompson (1985) the modularization of the linguistic component makes the required effort to change natural language minimal. This is especially true for systems where the output consists of simple lists and tables. In these cases, the response generator needs very little customization.

The main reason for using modules is that this increases the ability to *extend* the system in various ways. Allen et al (1998) claim that collapsing the various modules together for an application makes the system hard to construct and debug, and hard to modify for new tasks and domains.

Extensibility[2] spans over natural language (NL) extensibility, portability, scalability and transportability. These terms are used in different ways by various authors. The view in this thesis is that *NL extensibility* refers to changing the natural language (e.g. from Swedish to English), *portability* refers to changing domain (e.g. provide a new knowledge base) whereas *scalability* refers to extending an existing domain (by adding new facts to the same domain), and *transportability* implies changing computational platform (e.g. from Unix to a Windows platform).

The need for extensibility in general, and portability in particular arises because the current capabilities of natural language dialogue systems are limited to narrow domains and restricted contexts (Hafner & Godden, 1985).

The modular approach is suitable when dealing with extensibility issues. Modules with clear-cut borders can be reused, and ported to new modules. The two main modules are in most cases one linguistic module, and one background management module. To separate the linguistic component from the background system implies that the general linguistic framework can be reused when extending the system. The same

[2] Extensibility is sometimes refered to as *customization*. The two terms are used interchangeably in this thesis.

goes for the background system. With a generic and separate domain knowledge manager, the system is more easily ported to a new domain. The dialogue manager should only be involved in processes concerning the actual dialogue with the user, and not be involved with background system access. The domain knowledge manager should, according to Flycht-Eriksson and Jönsson (2000), handle this. Dahlbäck and Jönsson (1999) conclude that a modular approach makes it possible to reuse the core of a dialogue system when extending it to a new domain, without having to incorporate aspects of dialogue management that is not required in the new domain.

To customize an existing system to a different platform, Thompson and Thompson recommend using *open systems standards*. The consequences of open systems standards are that components can be implemented on different hardware by different vendors, making them more transportable and extensible. Development time and cost can be minimized since many aspects of a system can be built of existing components, and be adapted to evolving needs. Examples of open systems are the programming language Java and the Linux platform. This is to contrast with for example the proprietary Allegro Lisp compiler.

The process of analyzing syntax and semantics in text is called *parsing*. Before describing parsing in further detail (see section 2.4.3 Chart Parsing and 2.4.4 Lexicon and Grammar), explanations of feature structures, unification (section 2.3) and two grammar formalisms (sections 2.4.1 Context-Free Grammar and 2.4.2 Unification-Based Formalism) are needed.

2.3 FEATURE STRUCTURES AND UNIFICATION

Two central concepts of computational linguistics are *feature structures* and *unification*. In general, they belong to a reductionist approach that allows explanations of a large structure's behavior as a function of the combined behaviors of its substructures (Jurafsky & Martin, 2000). A structure can be thought of as representing an object, which can have complex sets of properties attached. Models that use such structures are called *constraint-based formalisms*.

When implementing a constraint-based formalism, an attribute-value matrix can be used. This is referred to as a feature structure. Figure 2 shows an example of a simple feature structure:

$ATTRIBUTE_1$	$VALUE_1$
$ATTRIBUTE_2$	$VALUE_2$
...	...
$ATTRIBUTE_n$	$VALUE_n$

Figure 2 A simple feature structure with atomic values, represented in an attribute-value matrix.

The value fields in Figure 2 consist of atoms, but they can also contain substructures, as in Figure 3.

$ATTRIBUTE_1$	$ATTRIBUTE_{1a}$	$VALUE_{1a}$
	$ATTRIBUTE_{1b}$	$VALUE_{1b}$
$ATTRIBUTE_2$	$VALUE_2$	

Figure 3 A feature structure containing a substructure.

A feature structure can be represented in a graph, which illustrates a feature path. This is called a *Directed Acyclic Graph* (DAG) and is shown in Figure 4.

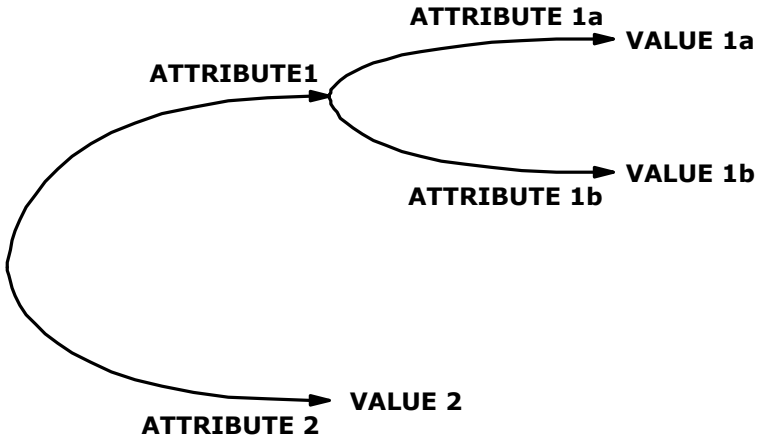

Figure 4 The feature structure represented as a Directed Acyclic Graph (DAG).

A third way to represent feature structures is in equation form, as shown in Figure 5. This makes them, as we shall see, suitable when using feature structures in conjunction with context-free grammatical rules (see section 2.4.2 Unification-Based Formalism).

$0\ \text{Property}_1\ \text{Property}_{1a} = \text{Value}_{1a}$:

$0\ \text{Property}_1\ \text{Property}_{1b} = \text{Value}_{1b}$:

$0\ \text{Property}_2 = \text{Value}_2$.

Figure 5 The feature structure represented as an equation. '0' is a reference to the top of the structure.

Feature structures can be used to represent information about an object and its properties. In language technology, feature structures are particularly useful for both parsing (see section 2.4.3 Chart Parsing) and domain reasoning (see section 2.5.2 Reasoning About Domain Knowledge).

The main advantage of using feature structures is that several structures can be compared and merged through *unification*. A unification procedure compares two feature structures and returns – if possible – one unified

structure. Figure 6 shows a successful unification of two feature structures.

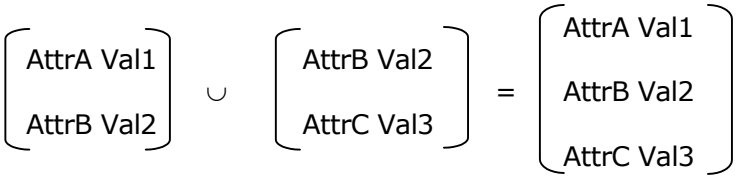

Figure 6 Unification of two feature structures.

The unfication results in the creation of a new feature structure containing the union of the information from the two original feature structures. A complete unification algorithm is presented by Jurafsky and Martin (2000, p. 423).

2.4 GRAMMAR FORMALISMS AND PARSING

There are a number of different grammar formalisms that can be used in dialogue systems for natural language. To provide the background information necessary for the concept of chart parsing, two grammar formalisms are discussed. First, context-free grammar, which builds on categories that are ordered according to a set of rules, is described. Second, the unification-based formalism that combines context-free grammars with unification of feature structures is explained.

2.4.1 CONTEXT-FREE GRAMMAR

The idea of context-free grammars (or phrase-structure grammars) was formalized by Chomsky (cf. Jurafsky & Martin, 2000) and consists of a set of rules for predefined categories. Each rule expresses the way that the categories can be grouped and ordered together. Figure 7 shows a simple context-free grammar:

S → NP VP	Abbrevations:
NP → N	S = sentence
NP → PROPN	P = phrase
VP → V NP	N = noun
N → Linguistics	V = verb
PROPN → Abraham	PROPN = proper noun
V → likes	

Figure 7 Sample context-free grammar.

The sentence "Abraham likes Linguistics" is parsed as being a sentence belonging to the category S, since it consists of a noun phrase (category NP) with the proper noun (category PROPN) "Abraham", and a verb phrase (category VP) consisting of the verb (category V) "likes" and the noun (category N) "Linguistics". This can be represented in a *parse tree*, as in Figure 8.

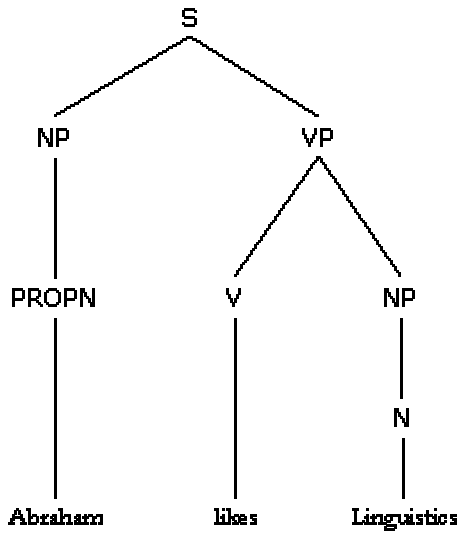

Figure 8 A parse tree generated from a context-free grammar.

Usually, context-free grammars are thought of as a device for assigning a structure to a given sentence, or as a device for generating sentences (Jurafsky & Martin, 2000).

2.4.2 UNIFICATION-BASED FORMALISM

Unification-based formalisms are based on Chomsky's transformational grammar. The drawback with this approach is that it is hard to construct efficient parsing algorithms. To come to terms with this, various unification-based formalisms were developed. Among them is the PATR formalism (cf. Strömbäck, 1996). PATR can be viewed as a context-free grammar with feature structures (i.e. DAGs) as categories. This means that the categories need not be atomic, but consist of features with attached values.

The use of feature structures allows for unification (as mentioned above), and is helpful when we want to merge partial information structures. This means that new information can be added to the structure as long as it is compatible. Figure 9 shows a sample PATR rule.

```
NP → DET N :
        0 head = 2 :
        1 numb = 2 numb :
        1 gend = 2 gend .
```

Figure 9 Extending context-free grammar with PATR unification constraints. From Hansen (1998, p. 2).

The first row is an ordinary context-free grammar rule. What follows is the DAG in its equation form (see section 2.3 Feature Structures and Unification). The DAG contains information about how to unify the categories.

Extending the context-free grammar with unification constraints, allows us to largely ignore the implemented search strategy used by the parser module. This is due to the order-independent nature of unification (Jurafsky & Martin, 2000).

Table 2 summarizes how context-free grammars, parse trees, feature structures, DAGs and unification relate to each other.

Table 2 Relationship between context-free grammars, parse trees, feature structures, DAGs and unification.

Information representation	Definition	Generation
Parse tree	Context-free grammar	Parser
Feature structure	Rule + DAG in equation form	Unification

The main point this leads to is that feature structures are more semantically orientated than parse trees, since the objects in them are described with respect to their *meaning* instead of their *order*.

The tools provided above are useful in the process of automatically parsing written natural language. The next section briefly describes parsing in general, and chart parsing in particular.

2.4.3 CHART PARSING

As mentioned earlier, *parsing* is the process of assigning part of speech attributes to each word in a sentence (Russel & Norvig, 1995). Usually, this denotes automatic analysis of a text, with a given lexicon and set of grammatical rules. The purpose of a parsing module is to interpret natural language user input, and generate a data structure to represent it.

Traditionally, complete syntactic analysis has been favored. For information extraction systems, care must be taken before trying to implement complete analysis. The problem is that it is too demanding computationally, and requires a lot of "unnecessary" work, when implementing the grammar and lexicon. Cowie and Lehnert (1996) advice that one can "get by with minimal syntax" (p.85), when implementing the

parser. This is referred to as *shallow* or *partial* parsing and is useful when extracting information, since the complete syntax is often not needed (Jönsson, 1993). Analysis in shallow interpretation is done by parsing as small parts of the utterance as possible (Jönsson & Strömbäck, 1998).

Related to shallow parsing is the parser's *robustness*. Since it is desirable to minimize the size of the lexicon, chances are that the user will use words not covered by it. With a robust parser, these pitfalls can be avoided. A parser that allow for *general rules* has the ability to leave some words non-analyzed. The main advantage of this is that the size of both the lexicon and the grammar can be decreased (Hansen, 1998). The notion of robustness forms one part of the attempt to fulfill Norman's (1988) requirement of error handling in a system with no constraints (see section 2.2.1 Natural Language Interfaces), and is one step in the direction of creating a habitable dialogue system.

The basic algorithm used for chart parsing is the *Earley* algorithm, which was constructed by Earley (1970). The core of this algorithm is a left-to-right pass that fills out an array called a *chart*. For each word in the parsed sentence, the chart contains information about the parse trees generated so far (called *states*). The function of the chart is to prevent repeated parsing of the same string, and to prevent that the same is generated more than once. This is done through prediction, scanning and completion (Jurafsky & Martin, 2000).

Chart parsing is object oriented. Each meaningful text unit (depending on the rule that is used when parsing text) corresponds to a hierarchically organized object. This makes it suitable not only for syntactic and morphological analysis, but also for compositional semantic representation (Wiren, 1992). The interpreted input is represented as a feature structure in the matrix format.

2.4.4 LEXICON AND GRAMMAR

The parser utilizes a word list, called a lexicon. The lexicon should theoretically consist of all lexical terms occurring in the sub-language of the genre in which the system is supposed to be used (Jurafsky & Martin, 2000). In practice, this is not feasible, since it requires too much effort and computer power.

A problem when constructing lexicons in English is the notion of *open nominal compounds* ("ice cream", "action movie"), and *phrasal verbs* ("married [somebody] off"). The traditional view of a word as being a contiguous sequence of alphabetic characters is thus not completely true (Amsler, 1989). It is desirable that the lexicon format supports open nominal compounds and proper nouns (such as first name and last name); otherwise this has to be implemented in the grammar rules.

The simplest – but largest – possible lexicon would be a list of all existing words in a natural language (or sub-language). This would be a very large and cumbersome list, which would not reflect the morphological structure of the language, and the dynamic aspects such as derivations and compounds. Also, the list would contain needless duplicates of morphemes. To come to terms with this, a *computational lexicon* – or mini-lexicon – is recommended by Jurafsky and Martin. The computational lexicon consists of a list of stems, and universal morphological rules for the chosen natural language. The main advantage of this is the fact that all words belonging to a certain word type behave the same can be used when building the lexicon. For example, all nouns ending with a '-y' in singular (e.g. "sky", "city", "wallaby") behave the same when transformed into plural. They all drop the '-y' ending and get the ending '-ies' ("skies", "cities", "wallabies"). Instead of listing all possible derivations from a word stem, the computational lexicon only contain the stem itself, along with information about what type of word it is. From

there, the various forms and syntactic categories originating from the stem can be derived.

The grammar in chart parsing follows a unification-based formalism and consists of rules that decides how the DAGs connected to the words in the lexicon are to be grouped and unified (see section 2.3 Feature Structures and Unification).

The main advantages of reducing the size of the lexicon and grammar are that development time, and computation time, are reduced (Jönsson & Strömbäck, 1998).

2.5 REPRESENTING AND RETRIEVING KNOWLEDGE

In the previous section the relevant computational linguistic framework has been discussed. Now it is time to deal with the background system, i.e. the knowledge base. The purpose of a knowledge base is to provide the system with information, and means to access and reason about it.

2.5.1 RELATIONAL DATABASES

A *relational database* represents a coherent model of a part of the world (Androutsopoulos et al, 1994). A relational database can be defined as a collection of information organized into interrelated tables of data and specifications of data objects. The information is structured in tables that are connected to each other through primary and foreign *keys*. This means that the information can be retrieved using a formal query language (e.g. SQL). A formal query language is artificial and requires training to master. In order to remove ambiguity and redundance in the tables, databases can be normalized. *Normalization* is the process of structuring a relational database schema such that most ambiguity is removed.

To make the information retrieval task easier for users not willing or able to learn a formal query language, two other interaction techniques have been designed: graphical interfaces and form-based interfaces. Graphical interfaces allow the user to use drag-and-drop techniques with a mouse to manipulate and extract data. Form-based interfaces use predefined fields that can be filled out by the user. These three approaches all have their own advantages and disadvantages, and the decision which one to use in a particular system depends on the complexity and type of task. They all have in common that they require some extent of training before they can be properly employed. One answer to the need for easier interaction techniques is the natural language interface, discussed in section 2.2.1 Natural Language Interfaces.

2.5.2 Reasoning About Domain Knowledge

As mentioned in section 2.3 Feature Structures and Unification, feature structures and unification-based formalisms are useful for semantic information representation. This makes reasoning about domain knowledge possible. Flycht-Eriksson and Jönsson (2000) recommend a generic domain knowledge manager with various knowledge modules (e.g. temporal reasoning module, spatial reasoning module). When customizing the generic framework to a specific application it is advantageous if the domain knowledge manager is only concerned with phenomena related to the background system.

There are different *communicative acts* a user can perform when interacting with a dialogue system. Flycht-Eriksson and Jönsson define a structure shown in Figure 10 for different communicative acts.

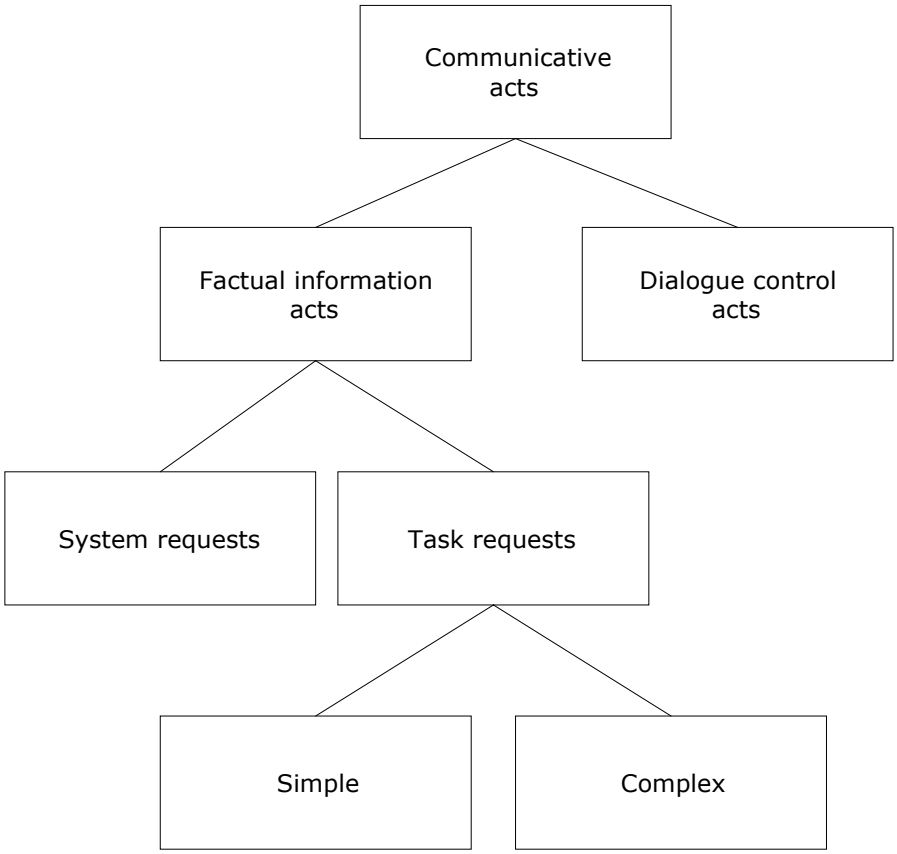

Figure 10 Communicative acts.

2.5.3 TEMPORAL REASONING

Temporal information is expressed in a number of ways in natural language. In order to objectively model temporal expressions the linguistic form needs to be analyzed. Smith (1981) categorizes temporal frame-adverbials into three categories:

- *Deictic phrases* (e.g. "last week", "tomorrow")
- *Dependent phrases* (e.g. "later", "after")
- *Clock calendar phrases* (e.g. "at midnight", "at noon")

Deictic phrases are anchored in the time of speech, and a dependent is anchored in another given time in the context, whereas a clock calendar can anchor to either speech of time, or another context-dependent time (Merkel, 1988).

Merkel makes a distinction between *periods* and *phases*. Periods are segments of time with a certain length and can have a number of subperiods that together exhaust the main period. A phase is a specific instance of an ordered subperiod.

Periods can be either *fuzzy* (such as seasons, and ”night”) or *bounded* (such as months, and weekdays). Fuzzy expressions present a problem when trying to model them. In order to determine for example at what time ”afternoon” turns to ”night”, interpretation principles are needed. Another troublesome aspect of temporal expressions is the ”next” expression. This is ambigouos in some cases. Depending on speech of time, the expression ”next Thursday” can mean either the closest following Thursday (speech of time +1), or the Thursday in the next week (speech of time +2).

Merkel discusses time adverbials, and claims that order is not always necessary within a temporal phrase. The following phrases have the same meaning and consist of the same meaning-bearing constituents:

- on Monday next week at 6 pm
- At 6 pm next week on Monday
- Next week on Monday at 6 pm

Interpretation principles are needed to resolve temporal expressions. The result of a successful interpretation is, according to Merkel, a DAG with a set of phasal values.

2.6 NATURAL LANGUAGE GENERATION

Natural Language Generation (NLG) is when a speaker uses knowledge about a language to decide what to say (cf. Russel & Norvig, 1995). Artificial NLG has not been stressed as much as interpreting natural language within computational linguistics and artificial intelligence.

According to Russel and Norvig, this is because "humans are anxious to talk to machines, but are not as excited about them talking back" (p.657).

A specific type of NLG is *response generation*. This is the generation of the response to a query in for example an information extraction system. There are three classes of generation, ranging from simple to advanced:

- *Canned text*
- *Templates*
- *True generation*

Which of these methods to use depends on the system complexity, whether or not the natural language is going to change, and how big the range of the responses is. For information extraction systems, where the responses mostly consists of tables and lists, canned text or templates are sufficient (Thompson & Thompson, 1985). Templates are based on the corpus, and are filled with relevant information from the knowledge base, depending on the request. Templates can be recursive and combined (Santamarta, 2001). True generation is expensive and requires tedious configuration (Androutsopoulos et al, 1994) and is only worthwhile if the output requires generation of complex sentences. It is important to note that even when dealing with simple task requests that give responses in the form of tuples from a database, additional types of responses are needed. Androutsopoulos et al present four cases where simply returning a database table is insufficient. They can be summarized as follows:

- The retrieved database tuples may contain encoded information that needs to be explained to the user.
- The system may not be able to understand the question, in which case no tuples are returned. The cause of failure should be explained to the user.

- The user's question may contain false presuppositions about the database or the world, in which case the system should report the false presuppositions.
- The user's request may not express literally what the user wants to know (e.g. Yes/No questions might require more than just a single "yes" or "no").

In attempting to provide appropriate error handling there are some system-related responses that need to be configured. Examples of this are when the user uses *modal questions* (i.e. questions about the relations between information in the database), or *meta-knowledge questions* (such as "Can I use slang when interacting with this system?"). Meta-knowledge questions are also referred to as system requests (as in Figure 10).

3 Problem Statement

This chapter consists of a development of the purpose, and forms questions that are to be answered in this thesis.

The purpose of this thesis is to explore how much and what kind of work that needs to be carried out when customizing a generic dialogue system framework to a new natural language, a new domain, and a new computational platform.

As pointed out in the previous chapter, a natural language dialogue system can be divided in two components: the linguistic component and the domain – or knowledge area – component. The same division can be made when breaking down this thesis' purpose. Thus, the purpose is viewed from two aspects: one linguistic aspect and one domain aspect. Each aspect has its own issues, and amounts to research questions. The focus of this thesis is to resolve some of the issues in the linguistic and domain aspects, the connection between them, and the workloads in the design and implementation of a dialogue system prototype.

The following sections give an overview of the various aspects of the purpose, and are concluded with the research questions of this thesis.

3.1 THE LINGUISTIC ASPECT

The linguistic aspect raises questions about what sub-language is being used by the to-be users. It is interesting to identify the sub-language, and find a suitable representation for it.

A second question of the linguistic aspect is how to increase the dialogue system's interpretation robustness. Depending on how well the sub-language has been identified and accounted for in the dialogue system, more or less robustness is needed. In all cases, since there are no constraints in the interface, the user will sooner or later provide input that has not been explicitly accounted for. A robust dialogue system will be able to accommodate this. Therefore, it is desirable to find out how a dialogue system can increase its robustness.

The generic framework, as well as the chart parser module (see chapter 4 The MALIN Framework), was developed for Swedish. In this thesis, the natural language is English. This raises another linguistic question: what needs to be done when changing the natural language in the interpretation and response generation?

To summarize, this thesis deals with three questions raised by the linguistic aspect of the general problem:

- How is the sub-language identified, and how is its representation defined?
- How do we enhance interpretation robustness on utterance level in a dialogue system?
- What needs to be done when changing natural language in a dialogue system?

3.2 THE DOMAIN ASPECT

The knowledge area for this thesis is the realm of television programmes and television programme tableaus. There exists a wide range of channels, which broadcast an even wider range of programmes. The programmes have titles, start times and start dates; they are divided in categories, certain programmes have actors, directors and presenters, etc. The information available about television programmes is not only vast; it is also subject to change. Every day, new shows are created and broadcasted, so even though the domain is set, the content changes constantly.

On the domain level, several interesting and challenging questions require attention. As with the sub-language, the domain has to be identified since there are not enough resources to cope with a generic cross-domain system. Just as with the definition of the sub-language representation, the domain information representation is crucial to define.

The analogy with the linguistic aspect continues with the extensibility issue. The generic framework described in chapter 4 The MALIN Framework needs to be customized to a new domain. This is referred to as the system's portability, and finding out what characteristics make the system more or less portable is relevant, since this in turn affects the system's scalability. Scalability is a special case of portability and refers to the system's ability to upscale – or change content – within an already existing domain. Identifying how to make allowance for scalability is especially interesting for dialogue systems that are dealing with an ever-changing domain, such as the television tableau domain. This implies design decisions in data representation, as well as system architecture.

Three main questions are posted in this section:

- How is the domain identified, and how is the representation of the information in the domain defined?
- How is portability in a dialogue system enhanced?
- How do we construct a system architecture and data structures that allow for scalability?

3.3 COURSE OF ACTION

Having broken down the general problem into linguistic, domain and workload aspects and posed a number of questions, the course of action has to be set. To contribute to the issues described above, a functional prototype of a dialogue system, based on the MALIN architecture, is to be designed and implemented for the television tableau domain. Dialogue systems have been developed before, and the prototype developed in this thesis is not particularly advanced. The aim of the development is to complete a full development iteration to find out where the main efforts in a more full-on, robust dialogue system development should be put.

4 The MALIN Framework

Before moving on to the design and implementation of the dialogue system, the general architecture of MALIN, and its predecessor LINLIN, needs to be described. This architecture is a generic dialogue framework, which design contributes to the system in this thesis. MALIN and LINLIN are developed at NLPLAB. Along with the MALIN architecture, a chart parser called Flexchart, is provided.

4.1 THE MALIN SYSTEM ARCHITECTURE

The LINLIN architecture is described by Jönsson (1993; 1997) and is a natural language dialogue system framework originally written in Lisp. It is the predecessor of the multimodal MALIN architecture, currently used by NLPLAB (Degerstedt & Jönsson, 2001). The LINLIN framework has previously been implemented in Swedish on a Unix platform in a used cars domain called CARS (Jönsson, 1993). The approach taken in LINLIN is to let the dialogue manager be the central controller of the interaction. The architecture of the CARS customization is shown in Figure 11.

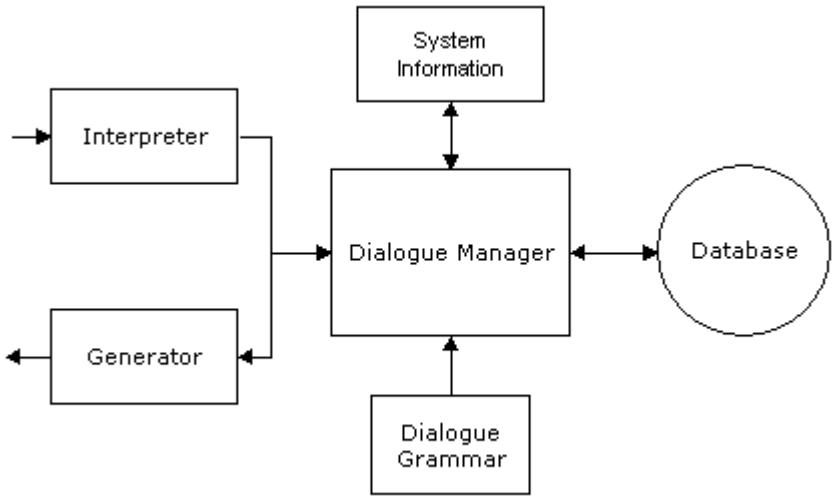

Figure 11 The architecture of the LINLIN-based CARS dialogue system.

A new architecture has been suggested to enhance the domain reasoning capabilities of the LINLIN framework. The architecture and connection to other modules of the new framework has been named MALIN and is shown in Figure 12 (Flycht-Eriksson & Jönsson, 2000). The MALIN framework is written in Java.

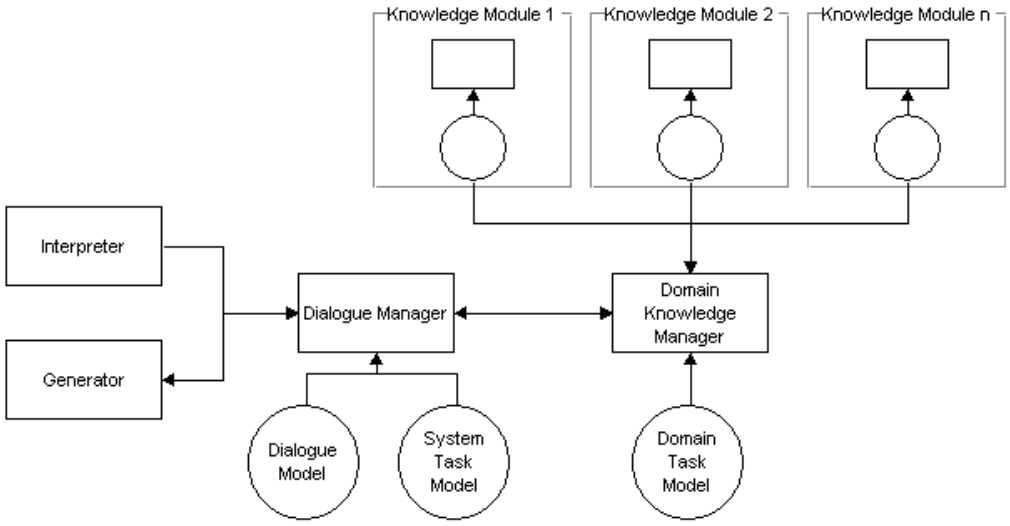

Figure 12 The conceptual design of the MALIN framework (from Flycht-Eriksson, 2000).

MALIN is the architecture that is to be customized for the television tableau domain in this thesis. The starting point for the system in this thesis is the CARS customization, which had no implemented domain knowledge manager module (see Figure 11). Currently, work is carried out at NLPLAB to implement the domain knowledge manager module for the MALIN architecture (Flycht-Eriksson, 2001), but at the time of writing, that work is not finished.

Jönsson (1993) identifies two important concepts for simple information requests: *objects* and *properties*. Simple requests deals with properties of and relations between simple objects, for which the answers can be values of properties or names of objects. A central concept of the object/properties approach is that they can be represented as feature structures, and be unified. The feature structures contain information about objects and properties belonging to them. Also, the structures can contain *markers*. A dialogue system and other modules reason about these structures. Henceforth, these structures are referred to as *OPMs* (Objects, Properties, and Markers) and they are a part of the LINLIN, as well as the MALIN

general framework. An object typically includes one or more fields by which the object is identified in the knowledge representation structure. The object-fields are thus related to the concept of key fields in relational databases. Properties have aspects and atomic or non-atomic values attached. The properties are aspects of the current object that the user and system are interested in. Markers can be viewed as flags that are used on a conceptually higher level than properties and object-fields. For instance, a marker flag can signal that the current OPM is referring to a system question – as opposed to a task-related query. Also, a marker can contain dialogue information, e.g. "greeting", or "farewell".

The MALIN architecture supports sub-language representation as unifiable OPMs. The domain specific OPMs are further described in section 5.4.2 The Dialogue Manager Module. Generally, they consist of objects, whose properties are modeled as aspect/value matrices. Figure 13 shows an example OPM from the CARS implementation:

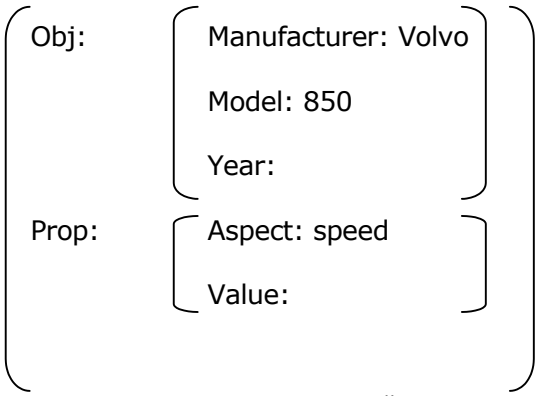

Figure 13 An OPM for the question "How fast is a Volvo 850" (from Flycht-Eriksson, 2001).

MALIN requires an interpreter that parses the user's input. In previous implementations the Lisp-based chart parser Flexchart, also developed by NLPLAB, has been used. MALIN communicates with Flexchart using a socket connection.

4.2 THE CHART PARSER FLEXCHART

Flexchart is a robust chart parser written in Lisp for the Unix platform, and has been used in the customization of the MALIN framework in the used cars domain. The original algorithm is from O'Shea and Eisenstadt (1984), but has been modified since (Hansen, 1998). Flexchart is robust, because it allows for partial parsing. The means for doing this is to construct general rules in the grammar. In Flexchart, an important aid to construct general rules is the use of a "Kleene star" (i.e. the * symbol), to denote *any* word. The main advantages of this are that the size of the grammar decreases, and the parsing gets less prone to errors such as typos, and makes up for lack of lexicon coverage.

The grammatical formalism in Flexchart is the unification-based PATR (see section 2.4.2 Unification-Based Formalism), which means that the rules can contain DAGs (in equation form) that can be unified. Flexchart takes a text string as input and produces a feature structure in string format as output.

Flexchart is flexible, since it can be loaded with a grammar and lexicon of choice. The grammar and lexicon are implemented as two separate text files. The lexicon supports the use of a computational lexicon, where the language-dependent morphological information is contained in a third separate text file.

Hansen (1998) claims that since the parser in itself is not an executable file, the parser is not transportable.

5 The TV-Programme Dialogue System

In order to resolve the questions posted in chapter 3 Problem Statement, a functional prototype of a natural language is implemented, which is presented in this chapter. First, the relevant method theory is described in section 5.1 Method. Section 5.2 Corpus Work gives an account of the corpus work. The two following sections (5.3 Design and 5.4 Implementation) describes the design and the implementation of the dialogue system. Finally, to clarify the system flow, a detailed walkthrough of an example query is presented in section 5.5 The System Flow: An Example.

5.1 METHOD

The implementation of a dialogue system for a new application can – according to Degerstedt and Jönsson (2001) – be divided in two distinct steps: *conceptual design* and *framework customization*. Design and coding are viewed as complementing each other in the process of system development, and are carried out in parallel. The conceptual design results in a design document. Working from the other angle results in the actual implementation.

This method is iterative. After a first prototype of the system has been created, the design and corresponding modules are updated, and a new

version of the system is created. One important factor in an iterative approach is that the realization process can be divided into manageable pieces. Furthermore, new insights and aspects occur, as the understanding of the problem increases. Step-by-step refining of the system helps to reach the overall goal with a future robust and habitable natural language dialogue system. This section describes the design and implementation of a first iteration. Chapter 7 Discussion discusses the obtained results, which provide the basis and guidelines for the next iteration. (In Figure 14, this is represented as the "Testament" box).

In summary, the design and implementation process can be viewed as a work chart. Figure 14 shows the workflow for the first iteration of the dialogue system described in this thesis.

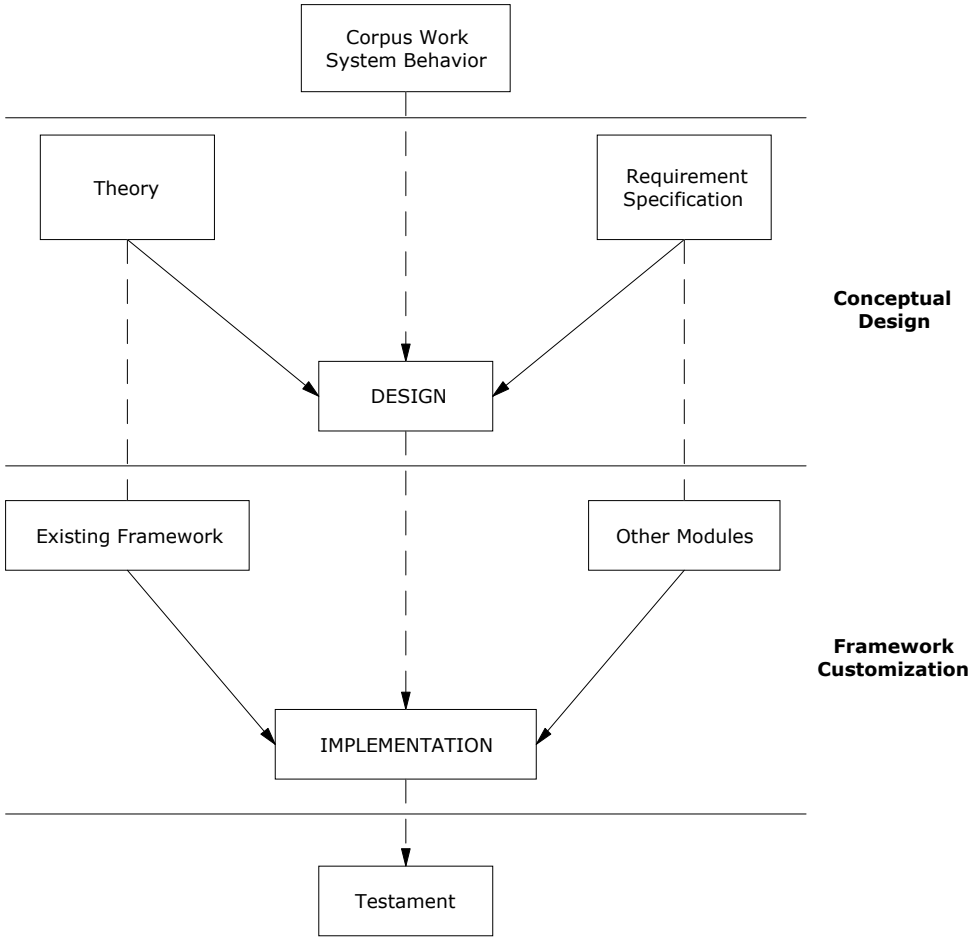

Figure 14 The system development work chart for the first iteration. From Degerstedt and Jönsson (2001, p. 2).

Degerstedt and Jönsson's notion of *prerequisites* before conceptual design and coding can commence consists of choosing theories, specifying the requirements of the system, and gathering and analysis of a dialogue collection (see section 5.2 Corpus Work).

The requirements specification can be summarized as follows:

- The interaction is typed, and in natural English.
- The area of knowledge is limited to television programme information.

- The system runs on a Nokia Mediaterminal compatible platform (i.e. Linux).
- There is no dialogue handling implemented.
- The communicative acts supported are simple task requests (e.g system requests are not accounted for).
- The MALIN generic framework forms the basis of the customization.

The complexity of the system is between the first two levels in Table 1 (chapter 2 Related Research), since the system does not provide clarification questions, but does allow the user to take initative.

5.2 CORPUS WORK

In order to determine what behavior the system should exhibit, and how broad the linguistic and domain coverage should be, a collection of dialogues is used. The analysis of the corpus forms the basis for the specification of the system. There is also a second corpus available, due to the introduction of the database content. This corpus is mainly used for including domain-specific words, such as programme titles, channel names, categories, and actors etc. in the lexicon. For the purpose of grammar development, the dialogue collection was used.

The dialogue collection consists of utterances that the system should be able to answer. For the first iteration this corpus was constructed through introspection, and consists of 50 utterances. These utterances, along with an addition from an informal test (see section 5.1.5 Updating the Corpus), represent the use cases of the system. Two analyses of the corpus were done. The first analysis aimed to divide

each utterance into meaning-bearing phrase chunks. It was revealed that the utterances consisted of sub-phrases (or components). Following the principle of compositionality (see section 2.2.2 Linguistic Coverage), it is assumed that the meaning of the utterance is equal to the sum of the meanings of the sub-phrases of the utterance.

The meaning-bearing components of the sentences in the corpus are of the following types:

- *Introduction Phrases*
- *Query Phrases*, consisting of up to three Query Blocks
- *Temporal Phrases*

The minimal utterance consists of a query phrase with one query block, but can contain all of the components.

5.2.1 INTRODUCTION PHRASES

The introduction phrases bear little relevant information. In the project corpus there are five types of introduction phrases.

- **Type I-1**: The user wants to know if something exists, by starting the sentence with phrases like "is there" and "are there".
- **Type I-2**: These include commands such as "show me", "tell me", "explain" etc. They can occur after courtesy phrases such as "please", "could you" and combinations (e.g. "Could you please tell me", "please explain").
- **Type I-3**: These phrases contain the words when/what/where/which/how etc, and can also contain an appropriate verb (e.g. is, does, has).

- **Type I-4**: This opening is used when the user explains what he/she wants to know. Examples are "I would like to know", "I want to watch", "I want to". There is a variation on this, which is a negation such as "I don't". These are rare, and mostly they deal with system questions.
- **Type I-5**: This type is related to the wish type above, but is formulated a little differently. "Can I", "can my VCR", "can I print", "can I watch" are all examples of this.

Introduction phrases are cumulative. It is possible to apply more than one introduction phrase in the same utterance. For example, the utterance "[Could you] [please] [tell me] [what] movies are on tonight" contains three type I-2 phrases and one type I-3 phrase. Brackets show the divisions between the phrases.

5.2.2 QUERY PHRASES

The query blocks contain information about the actual domain. At least one query block should be provided; otherwise the system will not know what to look for in the database. Depending on how specific the user is, more query blocks can be identified. There are three kinds of query blocks in the project corpus. Their order is somewhat important, but there are cases where the sequence can be omitted.

The query phrases are divided into three types. The main issue is not the type divisions though. Each query block contains one piece of useful information, or action word. That is what differentiates them. If the user provides three essential attributes or action words, three query blocks are filled out. The first query block contains one of the following types:

- **Type Q-1a**: Contains action words such as "shows", "programs", and "channel". They also contain phrases such as "movies are on", movies are showing" etc.

- **Type Q-1b**: Contains a set operating word and an action word. Examples are "any channel", "all movies", and "a news channel".
- **Type Q-1c**: Is closely related to Type Q-1a, since an Introduction Phrase of type I-3 often precedes them. They are more oriented towards a specific property (such as director, actor, description), instead of a more generic channel or category.
- **Type Q-1d**: This type is only found in one instance in the original corpus, and denotes a time phrase *before* the action or query block. The example is: "the next episode of". This could be viewed as one query block (of Type Q-1a) and one temporal phrase.

The second query block can be of five types according to the corpus. They are:

- **Type Q-2a**: On a specific channel (e.g. "on BBC2", "on Nature Planet").
- **Type Q-2b**: Persons (e.g. "with Demi Moore", "starring Mel Gibson", "directed by George Lucas").
- **Type Q-2c**: Categories (e.g. "feature cultural programmes", "shows sitcoms").
- **Type Q-2d**: Title of a programme (e.g. "Braveheart", "Gladiator", The Simpsons).
- **Type Q-2e**: The special case of including all of 1 above (e.g. "TV").

Only in two instances have the third block proven to be necessary. In the sentences "Tell me what Bravehart *is about*", and "Show all movies *currently screening*". The phrase "is about" (**Type Q-3a**) denotes the description aspect, and "currently screening" (**Type Q-3b**) denotes a temporal phrase ("currently") before the action word "screening". The reason to let these

special utterances represent a block of their own is necessary, since a request from the user might already consist of two query blocks. Table 3 shows how an utterance requiring three query blocks is composed.

Table 3 The components of an utterance from the corpus.

"Tell me	what	the action movie	on Channel4	tonight	is about."
INTRO 2	INTRO 3	QUERY 1B	QUERY 2A	TEMPORAL	QUERY 3B

5.2.3 TEMPORAL PHRASES

Analyzing the corpus gives an important fact about temporal expressions. The temporal expressions used in the corpus are often "sloppy" and vague descriptions of time that are not suitable for queries to a database that requires exact numbers for starting times and dates. The temporal expressions in the corpus form the basis for developing a time grammar. Section 5.4.3 The Domain Knowledge Manager Module describes the modeling of temporal expressions.

5.2.4 LOOKING FOR ANSWERS

The second corpus analysis categorizes the utterances on basis of what answers the statements should have the system generate. This forms the basis for connecting an information representation structure to the information extraction component. The corpus is sorted into categories depending on what information was provided and which facts that are expected. Classifying the questions into categories, or patterns, is important when it comes to generating the database queries. In total, 25 types were found in the corpus. Some of the basic question types from the corpus are listed in Table 4. (See also Table 8).

Table 4 Examples of question types in the corpus.

Given information	Expected information	Example
time	title, channel, time	"What shows are on television tonight?"
time, category	title, channel, time	"What movies are showing tomorrow afternoon?"
time, title	title, channel, time	"Is Star Wars screening this weekend?"
time, category, actor/director/presenter	title, channel, time	"Are there any movies directed by George Lucas tonight?"
time, channel	title, time	"What is on Channel 5 tomorrow after 6 pm?"
time, channel, category, actor/director/presenter	title, time	"Are there any movies with Nicole Kidman on BBC1 between 9 and 11 pm on Wednesday?"
title, actor/director/presenter	Y/N	"Is Ridley Scott the director of Gladiator?"

5.2.5 UPDATING THE CORPUS

The original corpus was used to produce a lexicon and set of grammar rules. Even though no formal tests have been conducted for collecting or

testing the corpus, a simple and informal test was conducted using one subject. The instructions were to freely type questions to the system using English. The use of polite phrases, synonyms and all kinds of time phrases were encouraged. All interaction was logged.

The test resulted in 35 new sentences, which were added to the corpus. This extended project corpus was used to construct a new version of the lexicon and grammar.

It was concluded that the interaction started out being simple and general, but got more complex and specific, since the parser seemed to handle all requests accordingly[3]. The first query was:

"What is on television tonight?"

The last query was more complex:

"Tell me who is starring in the movies on BBC2 that starts after five pm and ends before ten pm?"

5.3 DESIGN

The conceptual design of the system is done in parallel with the implementation (see Figure 14) and results in a design document, which specifies what sub-modules are included in the final system, and how the system should behave, as well as what kind of input is expected. The two latter goals are in general accounted for from the corpus work. The former goal, identification of sub-modules, is reached in this section. The design can be seen as providing guidelines for the implementation.

[3] This complexity increase might have been due to the eagerness to try the system's limits. It has no bearing on how the system might be used in a real context. The only feedback that was given to the user in this test was the returned chart parser feature structure.

5.3.1 KNOWLEDGE REPRESENTATION

By analyzing and drawing conclusions from the user-centered dialogue collection described in section 5.2 Corpus Work, as well as ”working from the other end” (i.e. considering the task-centered XML documents that provides the content of the database), the knowledge representation structure is defined. The structure is referred to as an *OPM* (see section 4.1 The MALIN System Architecture), and is a unifiable feature structure. The OPM is defined to contain one or more of the parameters listed in Table 5:

Table 5 Objects and Properties in the television system.

Entity	Field		Value
object	Channel		∞[4]
	Date		<YYYYMMDD>[5]
	Start		<HHMM>[6]
properties	aspect	actor	∞
		category	children culture film music news series sport talkshow
		description	∞
		director	∞
		presenter	∞
		subtitle	∞
		time	afternoon day morning night
		clock	noon midnight am pm
		day	today tomorrow Monday … Sunday
		title	∞
		year	<YYYY>

The fields CHANNEL, DATE and START identify an object, about which properties the user can ask. An important concept is that there are notions of time in both the object-fields (START and DATE) as well as in

[4] The infinity sign denotes that the value content can be a string of arbitrary length, consisting of alphanumeric characters.

[5] The format "YYYYMMDD" requires the dates to be represented as for example "20010303".

[6] Times are represented as for example "1025" for 10.25 am and "2225" for 10.25 pm.

the properties (with aspect CLOCK, DAY, and TIME). This makes temporal reasoning easier to manage (see section 2.5.3 Temporal Reasoning).

A specific OPM can contain everything from a single property with only an aspect, to a complex structure including all three object-fields and a hierarchy of properties with atomic and non-atomic values for each aspect.

5.3.2 MODULARIZATION

Identification of the system's modules includes specification of each module's responsibilities and the interfaces between them. This is necessary to make the development feasible and efficient. The modules in the design are outlined below (see Figure 15).

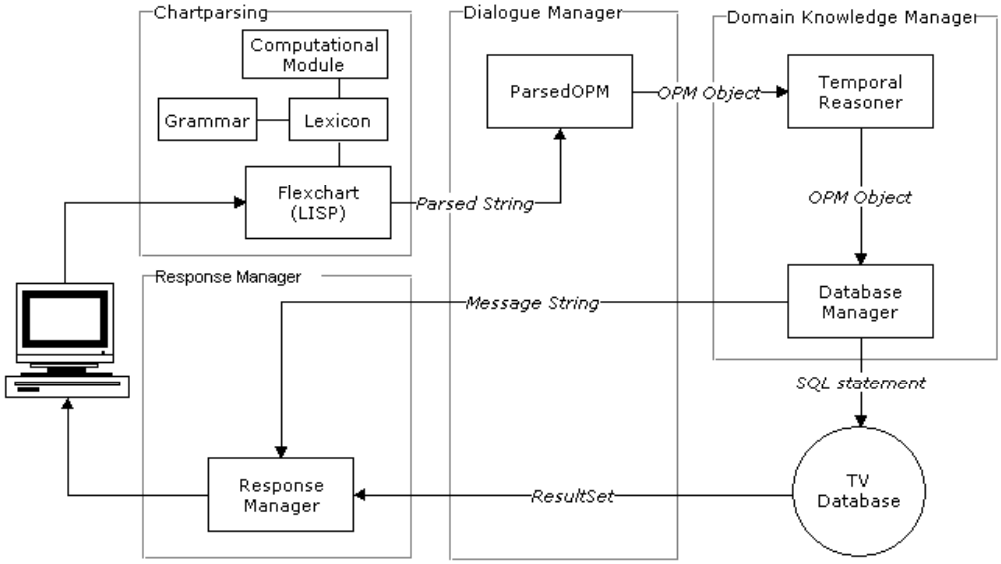

Figure 15 Conceptual design of the System architecture.

A **Chart parser** is needed to create a structure representing the user's input to the system. This structure is an OPM feature structure, and is produced by utilizing a domain-specific grammar and lexicon. A **Dialogue Manager** module keeps track of the dialogue history, and passes the information structure to a **Domain Knowledge Manager**

module. The Domain Knowledge Manager module utilizes a **Temporal Reasoner** to convert the user's temporal expressions to formalized notions of time that can be used in the creation of SQL templates. A **Database Manager** module that is contained within the Domain Knowledge Manager framework carries this out. The SQL statement created is sent to the **Database**, which contains the information the user wants. In order to present the results from the query, a **Response Manager** module formats and prints the system's output to the user.

This architecture is, as shown, straightforward and simple. The Dialogue Manager lacks some of the features that is desirable in the final version (e.g. a dialogue grammar to keep track of the dialogue tree and focus paramters). In this version, the responsibility of the Dialogue Manager is simply to be a placeholder, to which further functionality is to be added in later versions. The architecture above is minimalistic, which allows the user to post single task requests and get results back but little more than that.

5.4 IMPLEMENTATION

This section describes the implementation of the prototype, based on the design presented in the preceeding section. Each module is described in the following sub-sections.

5.4.1 THE PARSER MODULE

Two steps are involved when building the lexicon. First, the corpus is tokenized using a Perl script that separates all words and removes duplicates. The list is then sorted by syntactic categories. The result is a word list with 120 words, which form the basis of the lexicon. With a small corpus like in this case, the lexicon can be manually completed. For example, some weekdays were added, even though only "Monday" and

"Friday" actually occurred in the original list. Tokenization also includes the identification of lexical units, and abbreviations, which was done manually in this case. Some words are ambiguous, and need to be considered in the generation of lexicon and grammar (such as the verb and the noun "show"). First, a generic, static lexicon that is used in the system (independently of what actual titles; channels etc. are included in the database) is constructed. Second, a dynamic lexicon is created automatically from the database content. It is necessary to include all programme titles, channels, actors, presenters, directors and categories in the lexicon if the system is to understand a user's question that deals with explicit channels etc. The advantage of keeping two lexicons is that the dynamic lexicon can be exchanged on for example a daily or weekly basis. Both lexicons are loaded when the chart parser starts.

The identified components of the corpus utterances suggest that three separate grammars need to be developed. The framework for creating rules is provided by the corpus work (see section 5.2 Corpus Work). The introduction phrases are handled by one grammar, the query phrases by a second grammar, and the temporal phrases by a third grammar.

In this version of the system, the introduction phrases are implemented, but not accounted for in the OPM. The information contained in the introduction phrases has little use in the task-related queries. They could function as markers for dialogue issues, which is to be implemented in later versions.

The query phrases are of higher interest. The analysis of the corpus reveals that a query phrase can consist of up to three query blocks. To clarify, consider the following utterance:

"Which channel is Friends on tonight?"

The words in this string are looked up in the lexicon. For example, the word "Friends" belongs to the dynamic lexicon and its entry looks like in Figure 16:

```
friends = title (SHOW) :        0 lex = friends :
                                0 properties aspect = title :
                                0 properties value = friends .
```

Figure 16 Lexicon entry for the title "Friends".

Entries identify the grammatical *type*, the syntactic *category*, the lexeme, and the semantic information on how to order it in the OPM feature structure. The information used by the grammar rules is first of all that "Friends" is a TITLE. The information about ASPECT and VALUE are used when unifying with the rest of the words in the utterance. In the case of titles and proper names, the syntactic category is *semantically* oriented. In Figure 16, the syntactic type is SHOW. For other words (typically included in the static lexicon), the syntactic category actually *is* syntactically oriented. For example, the syntactic category for the word "film" is N1, meaning that it is a *noun* of type 1. The morphology interpreter uses this information when parsing different forms of "film" (e.g. "films"), (see section 2.4.4 Lexicon and Grammar). The parser uses the semantic information to produce a unifiable OPM feature structure.

The rest of the words are contained in the static lexicon. "Which" is for example of the constituent type WH (along with words like "which", "who", "when" etc.). The word "channel" is a generic instance of the type CHANNEL etc. The notation of the constituents is arbitrary, and consists of a mix between syntactic categories (such as V denoting any verb), and semantic categories (such as TITLE, indicating that the constituent is a title, regardless of its syntactic classification).

As shown in section 5.2 Corpus Work, utterances can be divided into Introduction, Query and Temporal phrases. Table 6 lists constituents and sub-phrases for the example utterance:

Table 6 Constituents and sub-phrases of an example utterance.

<table>
<tr><td>Utterance</td><td>"Which</td><td>channel</td><td>is Friends on</td><td>tonight"</td></tr>
<tr><td>Constituents</td><td>wh</td><td>channel</td><td>V title start</td><td>deictic</td></tr>
<tr><td rowspan="2">Sub-phrases</td><td rowspan="2">Introduction Phrase</td><td>Query Block 1</td><td>Query Block 2</td><td rowspan="2">Temporal Phrase</td></tr>
<tr><td colspan="2">Query Phrase</td></tr>
</table>

The meaning of the utterance is equal to the sum of the meanings of the sub-phrases, following the principle of compositionality (see section 2.2.2 Linguistic Coverage).

In the implementation, the parser applies the rules in Figure 17 to the example string:

```
1. Intro -> wh .
2. QueryBlock1 -> channel :                    0 object = 1 object .
3. QueryBlock2 -> V title start :              0 properties = 2 properties .
4. Time -> deictic :                           0 properties = 1 properties .
5. QueryPhrase -> QueryBlock1 QueryBlock2 :    0 properties = 1 properties :
                                               0 properties = 2 properties .
6. S -> Intro QueryPhrase Time                 0 properties = 2 properties :
                                               0 properties = 3 properties :
                                               0 object = 2 object .
```

Figure 17 Sample grammar rules.

The equations to the right in Figure 17 contain the necessary unification information. Categories are numbered from 0, so a '0' refers to the leftmost category, '1' to the first category after the arrow and so on. In the

example utterance above QUERYBLOCK2 is assigned the property information from the TITLE entry in Figure 17 (rule 3). That is to say that the meaning-bearing constituent in this instance of QUERYBLOCK2 is the title. Likewise, QUERYBLOCK1 is assigned the object-fields of the entry for the word "channel" (rule 2), and TIME copies the property structure from the DEICTIC according to rule 4, (in this case, "tonight"). The two query block structures are unified using rule 5 where the two property structures are unified. Finally, the Introduction, Query and Temporal sub-phrases are unified into one feature structure.

The output from the parser module is a feature structure that has been unified according to the rules in the equations in Figure 17. The feature structure is a simple string, presented in Figure 18:

```
[PROPERTIES:[2:[ASPECT:DAY
                VALUE:[ARG:TODAY
                       RELATION:THIS]]
             1:[ASPECT:TIME
                VALUE:[ARG:NIGHT
                       RELATION:THIS]]
             0:[VALUE:FRIENDS
                ASPECT:TITLE]]
 OBJECTS:[0:[CHANNEL: GENERIC]]
 TOPIC:TASK
 SET:OLD
 ANSWER:NIL]
```

Figure 18 OPM delivered by the chart parser for query "Which channel is Friends on tonight?"

The string in Figure 18 is passed to the Dialogue Manager module.

5.4.2 THE DIALOGUE MANAGER MODULE

The Dialogue Manager module has several responsibilities in a full-on dialogue system. In this prototype it is limited to creating a structure of

Java objects[7] of the OPM from the string passed on from the chart parser. The Java structure is laid out in the fashion showed in Figure 19.

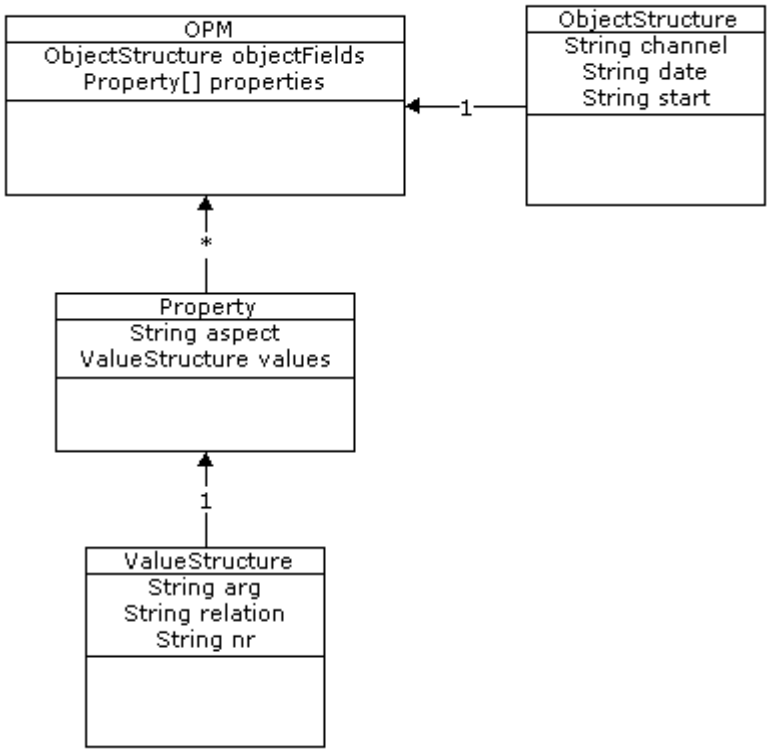

Figure 19 The OPM structure.

Each OPM contains one OBJECTSTRUCTURE with the three key fields CHANNEL, DATE and START. Furthermore, an OPM can contain any number ('*' in Figure 19) of properties (stored in the property array PROPERTIES). Each property consists of one ASPECT and a VALUESTRUCTURE spanning arguments (ARG), relations (RELATION) and numbers (NR). Each property is represented as a PROPERTY Java object. Relations and numbers are mostly used by properties with temporal aspects.

[7] A Java structure consists of Java objects, which are not to be confused with OPM objects. An OPM object corresponds to the Java class OBJECTSTRUCTURE, which can be instansiated to an OBJECTSTRUCTURE Java object.

The Domain Knowledge Manager uses the OPM to access various knowledge sources.

5.4.3 THE DOMAIN KNOWLEDGE MANAGER MODULE

The Domain Knowledge Manager module has two main responsibilities: resolving temporal aspects, and create a specific SQL statement from a set of generic SQL templates. Thus, as can be seen in Figure 15, the Domain Knowledge Manager module has two sub-modules: the Temporal Reasoner and the Database Manager.

The OPM created in the Dialogue Manager module normally lacks the object-fields START and DATE, but instead contains properties of various temporal aspects. In order to convert the often-vague temporal expressions provided by the user to a format that is compatible with the notions of time in the database the Domain Knowledge Manager module utilizes the Temporal Reasoner. As soon as a property with temporal information is detected, the Domain Knowledge Manager utilizes the Temporal Reasoner to convert the value, relations and number arguments to update the object-fields in the OPM. To clarify, consider the following user's input (Figure 20):

"What movies are on [before five tonight]?"

Figure 20 Example query.

The temporal phrase is enclosed in brackets. If today's date were March 3 2001, the corresponding and useful values to consider would be:

```
start < 1700

date = 20010303
```

These values are ready to be incorporated into an SQL statement that is sent to the database by the Database Manager. The Temporal Reasoner is

written in Java and takes properties with aspects TIME, DAY or CLOCK as arguments. The values of such properties can be arguments (TODAY, TOMORROW, PM, AM or NOON), relations (BEFORE, AFTER or AT) and/or a numbers (a point in time between 0 and 23). These properties translate to temporal frame-adverbials as in Table 7:

Table 7 Properties representation of temporal frame-adverbials

Frame-adverbial	**Representation**		**Example**
Deictics	Aspect: time	Value: arguments	*tomorrow*
Dependents	Aspect: time/clock	Value: relations	*before*
Clock calendar expressions	Aspect: clock	Value: arguments	*midnight*

The Temporal Reasoner works through the arguments and can produce two variables – either START or DATE. These values are in fact the object-fields of the OPM, which are updated with the variables. The returned variables are used when generating an SQL statement.

Below is the output from the Temporal Reasoner for the example input above. When the first property (CLOCK) has been parsed, the system does not know whether "5" refers to 5 am (i.e. 05:00) or 5 pm (i.e. 17:00).

```
PROPERTY: ASPECT: clock
          VALUE: [ARG:
                  RELATION: before
                  NR: 5]

(1) → start < 1700

(2) → start < 0500
```

Since the user did not provide a specific "am" or "pm" (i.e. the VALUE ARG is empty) the system does not know if the "5" provided should be 5 am or 5 pm. For now, options 1 and 2 are kept in memory.

Time is resolved when the next property is parsed since the value argument is "night". Today's date is used, since the user provided the word "tonight". The TIME property is resolved as follows:

```
PROPERTY: ASPECT: time
          VALUE: [ARG: night
                  RELATION: this
                  NR:]

→ start > 1800 AND date = 20010303
```

The heuristics of the Temporal Reasoner conclude that the missing ARG in the CLOCK property should be "pm", since the ARG of the TIME property is "night". This eliminates the early START option (2). It should be noted that the default value of "night" in the TIME property is interpreted as "later than 6 pm" ("> 1800"). The TIME property is overridden by the CLOCK property, so the final values returned by the Temporal Reasoner are:

```
start < 1700

date = 20010303
```

, which are the wanted values for the expression in Figure 20.

The updated OPM is now ready to be processed by the Database Manager. The Database Manager matches the included properties in the OPM and checks what – if any – fields are present in the object part. A query template, consisting of one SQL statement and one message string,

is created. The values and/or aspects in the properties and object-fields update the SQL statement's fields. The Database Manager has 25 query templates to choose from, and the choice depends on what information is present in the user OPM. Table 8 shows the matching between the templates and the information present in the OPMs.

Table 8 Matching between OPMs and query templates.

Query	Given information						Expected information						
	Object		Properties										
templ.	channel	start/ date	cate- gory	title	a/d/p	desc	title	channel	start	date	a/d/p	desc	cate- gory
1		x					x	x	x	x			
2		x	x				x	x	x	x			
3		x		x			x	x	x	x			
4		x			x		x	x	x	x			
5		x	x		x		x	x	x	x			
6				x	[x]		x				x		
7				x			x					x	x
8			x					x					
9	x	x					x		x	x			
10	x	x	x				x		x	x			
11	x	x	x		x		x		x	x			
12	x	x	x		[x]		x				x		
13			x		x		x	x	x	x			
14		[x]		x			x	x	x	x			
15		[x]			x		x	x	x	x			
16	x	[x]			x		x	x	x	x			
17	x	x		x					x	x			
18			x	x									
19	x		x				x		x	x			
20				x		[x]	x					x	x
21		x			[x]		x	x	x	x	x		
22	x	x			[x]		x	x	x	x	x		
23	[x]			x			x	x	x	x			
24	x		x		x		x		x	x			
25	x			x					x	x			

An 'x' indicates that a specific piece of information is present. An 'x' enclosed in brackets ('[x]') indicates that a property aspect but without value is present. This is also referred to as an "empty" value.

Continuing with the example above, Figure 21 shows the SQL statement from the chosen Query Template.

```
SELECT title.title, programme.channel, programme.start,
programme.date
FROM title, programme, category
WHERE title.ForeignKey = programme.PrimaryKey
AND category.ForeignKey = programme.PrimaryKey
AND category.category = "film"
AND programme.start < 1700
AND programme.date = 20010303
 ORDER BY programme.date, programme.start
```

Figure 21 SQL template with highlighted variable fields.

The bold lines show what has been updated. The Database Manager carries out the update using the information represented in the OPM, (after the Temporal Reasoner has updated it).

5.4.4 THE DATABASE

The data for the database was downloaded from an Internet site. The information was structured in XML and automatically converted to text documents. The text files were loaded into a relational database (see section 2.5.1 Relational Databases) that was set up on a MySQL server for the Linux environment. The Database Manager then loads a MySQL Driver and establishes a client connection to the database. The information in the database is distributed over six tables, connected through integers, that serves as artificial primary and foreign keys. Figure 22 shows the relationships between the database tables.

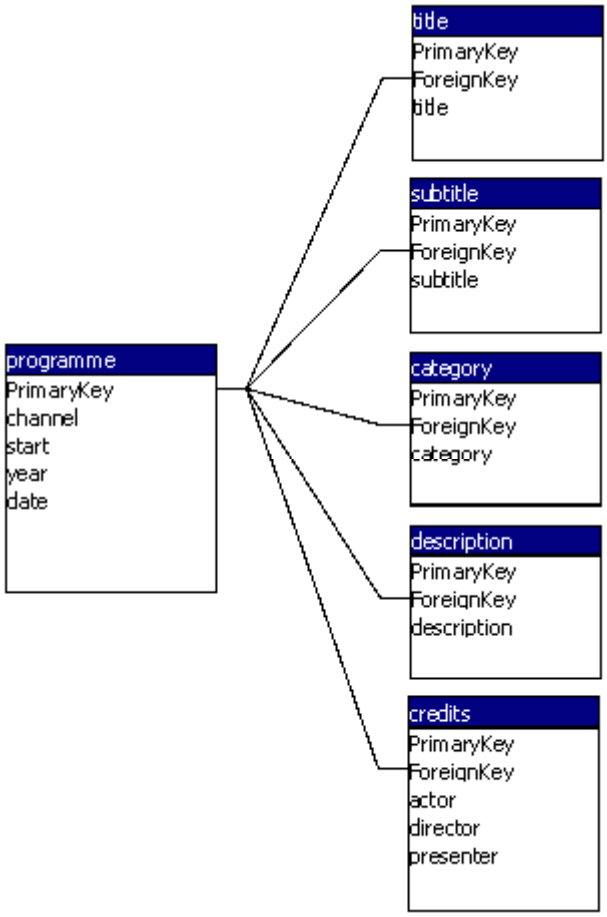

Figure 22 Relationships for the television database.

The tables in Figure 22 are normalized (see section 2.5.1 Relational Databases). In the demonstration version of the system the dates range from Saturday, March 3, 2001 to Friday, March 10, 2001. The database consists of programme listings for 34 channels during this period of time. This amounts to more than 4777 unique entries in the database.

In the main PROGRAMME table (to which the other tables are connected via foreign keys) the fields CHANNEL, START and DATE corresponds to the object-fields in the object feature structures. START denotes the starting time of the show.

The categories in the table CATEGORY are compatible with the category values in the OPM feature structure: CHILDREN, CULTURE, FILM, MUSIC, NEWS, SERIES, SPORT, and TALKSHOW.

5.4.5 THE RESPONSE MANAGER

The Response Manager has two main tasks: First, to format the resultset from the database and print it to the screen. Second, to customize a message template to provide a message string to go along with the resultset. There is one message template for each query template in the Database Manager, and is filled out using various aspects and values from the OPM.

When a resultset is retrieved, some tuples contain information that needs to be explained to the user. For instance, the values in the start fields are integers ranging from 0 (denoting midnight) to 2359 (denoting 11.59 pm). The start time '45 minutes past midnight' is for example encoded as '45', which is not an obvious point in time for the user. The first task (i.e. formatting) then consists of presenting values such as this in a readable format.

The customized message is categorized as a template (see section 2.6 Natural Language Generation). It provides two functionalities. The user gets confirmation that the system has in fact answered the right question, or – if it has not – gets information about how the system actually interpreted his or her question. Secondly, the message provides diagnostics if the resultset should be empty. Generally, it can be viewed as an error handling function, informing the user about empty resultsets, client-server time-outs etc. The system may not be able to understand the question, in which case no tuples are returned. The cause of failure is explained to the user by the message.

For Y/N questions, the use of meta-data (e.g. the number of tuples retrieved) along with the message is proven to be useful. Care is taken, since the user's request may not express literally what the user wants to know (e.g. Yes/No questions might require more than just a single "yes"

or "no"). In those cases, resultsets (if not empty) are reported as well. (see section 2.6 Natural Language Generation).

There is one message for each query template, and the message accompanies one unique SQL statement. The variables that are incorporated in the SQL statement can also be used in the message string. By using certain formatting rules, the messages can be customized for different locales. In English, the "am/pm" time format is preferred, while the most common time format in Swedish is the 24-hour "military time" format. To automatically switch between locales, a few simple parameters can be set in the Response Manager.

As an example, the message for query template 5 is provided in Figure 23.

```
The <creditsAspect> <CREDITSVALUE> is in the following <CATEGORY>
<date>:
```

Ex: The actor RUSSEL CROWE is in the following FILM(s) March 3, 2001:

Figure 23 Message string template for Query Template 5.

If the locale is set to for example U.S. standards, the format for <date> would be "March 7, 2001". In a Swedish locale setting, the format would be "7 mars, 2001".

The output from the Response Manager completes the chain of action in the system and provides the user with the system output.

5.5 THE SYSTEM FLOW: AN EXAMPLE

To summarize this chapter a complete walkthrough is presented. This section describes the flow of data from the user's input through all modules via the database to the system's output.

The user wants to know if there are any movies with the actor Rutger Hauer in the database. The user types the following sentence:

"Are there any movies with Rutger Hauer tomorrow night?"

This string is sent to the chart parser Flexchart via a socket connection. Flexchart processes the incoming string, utilizing the computational lexicon and the grammar, and delivers the following string (Figure 24):

```
[PROPERTIES:[4:[VALUE:[ARG:NIGHT]
               ASPECT:TIME]
            3:[VALUE:[ARG:TOMORROW]
               ASPECT:DAY]
            2:[VALUE:HAUER
               ASPECT:ACTOR]
            1:[VALUE:FILM
               ASPECT:CATEGORY]
            0:[VALUE:GENERIC
               ASPECT:CATEGORY]]
 OBJECTS:NIL
 TOPIC:TASK
 SET:OLD
 ANSWER:NIL]
```

Figure 24 The returned string from the chart parser.

The Dialogue Manager module uses various methods to create an OPM structure. Property 0 (the ”generic category”) is deemed less specific than property 1 (the ”film category”) and is discarded. The newly created OPM structure is passed to the Domain Knowledge Manager Module which scans for properties with temporal aspects. In this case, the Temporal Reasoner processes property 3 and 4 and updates the start and date fields in the object structure. The updated OPM is shown in Figure 25.

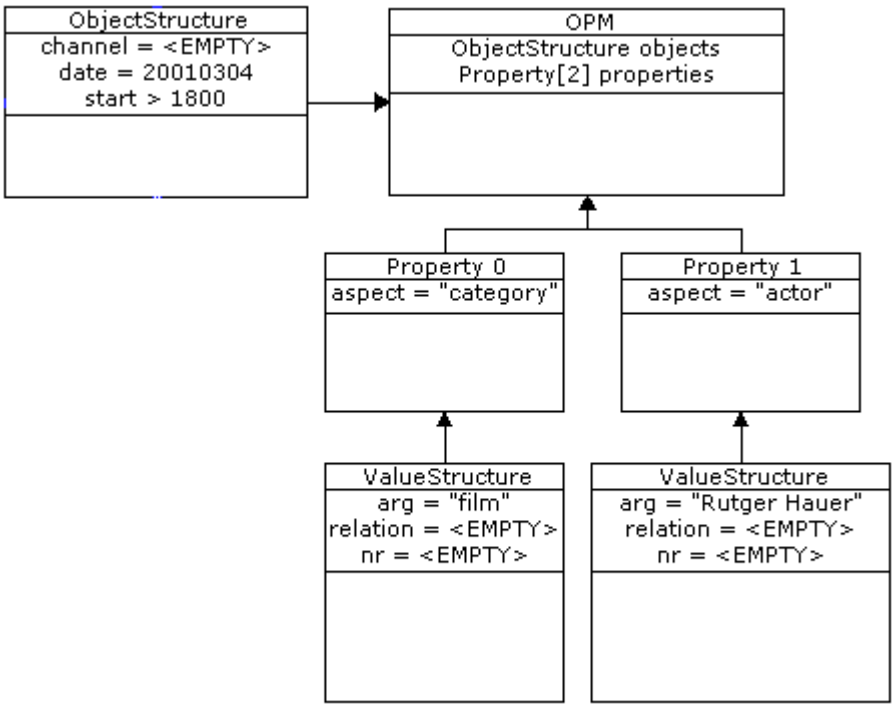

Figure 25 Example OPM updated with the Temporal Reasoner.

Now, the OPM is complete and the Database Manager tries to match the properties and object-fields to one of the 25 Query Templates. In this case the Database Manager recognizes the OPM structure and creates one SQL statement, and one customized answer string. Figure 26 shows the complete SQL statement with the incorporated CATEGORY, ACTOR, START and DATE variables.

```
SELECT title.title, programme.channel, programme.start,
programme.date
FROM title, programme, category, credits
WHERE title.ForeignKey = programme.PrimaryKey
AND title.ForeignKey = category.ForeignKey
AND category.category = "film"
AND credits.actor LIKE "%Rutger Hauer%"
AND credits.ForeignKey = title.ForeignKey
AND programme.start > 1800
AND programme.date = 20010304
ORDER BY programme.date, programme.start
```

Figure 26 SQL statement for the example OPM.

The Database Manager sends the statement in Figure 26 to the MySQL database, and passes the returned resultset to the Response Manager. The Response Manager formats the answer string and the resultset. The output is presented on the screen and looks like in Figure 27.

```
The actor RUTGER HAUER is in the following FILM(s) Monday, March
4:

Title: Omega Doom
Channel: Channel 5
Start: 10:00 PM
```

Figure 27 System output for the example query.

6 Results

The result of this thesis work is a dialogue system prototype. All experiences drawn from the corpus work, design phase and implementation phase are also results that are narrated in this chapter. The results are ordered by their contribution to the original problem statement in chapter 3 Problem Statement. First, the linguistic results are accounted for, followed by domain-related results.

6.1 LINGUISTIC RESULTS

The linguistic component of the system is concerned with three main aspects. The first aspect is how well the sub-language of the television programme domain has been covered for in the corpus, and how the representation form has managed to account for user requests (section 6.1.1 The Sub-Language and Its Representation). The second aspect involves the robustness of the linguistic component (section 6.1.2 Robustness). The third aspect deals with how the architecture handles the switch in natural language from Swedish to English (section 6.1.3 Natural Language Extensibility).

6.1.1 THE SUB-LANGUAGE AND ITS REPRESENTATION

Ideally, the corpus should be large and diverse enough to cover the complete sub-language of a given domain. In this case, the corpus size

and coverage is very limited. The corpus consists of 50 utterances, created mostly through introspection.

The domain span can be viewed from two angles. The first view is that the vocabulary of the television tableau domain sub-language consists of a relatively small set of words. The concepts in the domain deal with the classification of programmes; occurrences and durations of programmes; content and credits information about programmes (such as synopsis, actors, year of production etc); and information about channels and their content and broadcasting details. These concepts – except for temporal phrases – are all relatively easy to identify, and the sub-language can be viewed as static from this point of view, consisting of the words and expressions necessary to describe these concepts.

The second view of the sub-language is that it can contain an infinite range of words. Since the content of the television tableaus are constantly subject to change, there is a never-ending addition of for example titles, synopsis, actor names and channel names. This implies that the sub-language can also be viewed as dynamic.

Since this system is confined to task-centered queries, the communicative acts in the sub-langauge do not include meta-knowledge (or system) questions.

The sub-language in the MALIN framework is represented as OPMs. The exact layout of the OPMs is described in detail in sections 4.1 The MALIN System Architecture, and 5.3.1 Knowledge Representation. This form of representation works fine for the television tableau domain. What makes the feature structure representation functional is the design of the unification information in the grammar and lexicon. The chosen categories in the lexicon, and the division between introduction, query

and temporal phrases presents no significant problems in the implementation.

6.1.2 Robustness

The linguistic robustness often directly depends on the size and coverage of the project corpus. Since the corpus is limited, and the sub-language can be viewed as dynamic (see section 6.1.1 The Sub-Language and Its Representation), robustness is not easily achieved. In the prototype, utterance level robustness is handled by the general rules approach in the grammar. For example, the utterance "Is there a news channel" and the utterance (with a typo) "Is there a news channek" give the exact same output, since the general rule trigs on the keyword "news" to identify a category. Robustness on discourse level is normally handled by having a dialogue manager that collaborates with a domain knowledge manager. In the protoype, the dialogue manager is idling (i.e. discourse is not handled).

System robustness – to contrast with linguistic robustness – concerns the stability of the system while running and is discussed in section 7.3 Implementation Issues.

6.1.3 Natural Language Extensibility

To change natural language, both the input handler (the parser), as well as the output handler (the response generator) have to be modified.

The original chart parser was developed for the Swedish language. The lexicon and grammar are by necessity language dependent, and have to be rewritten. The only thing that obstacles a smooth transition to another natural language (such as English) is the computational lexicon module. If we are to use a computational lexicon to save development and computational time, we utilize the fact that a specific natural language behaves in a predictable manner for similar words. This implies that the

parser itself has to be modified when changing natural language. Fortunately, the computational module is implemented as a separate text file, following the same format as the grammar and lexicon, making the editing easy. Rewriting the computational module requires linguistic knowledge, but the module can be reused in various domains, as long as the natural language does not change.

The response manager has to present the results in the user's natural language. There are two ways for the system to respond to the user. The most obvious is the actual database content. The tuples returned by the database manager contains information of various kinds. Some information (e.g. such as titles, names of actors and directors) is language independent and can be presented as is. But some information (e.g. notions of times and dates) is dealt with in different ways depending on the language. The response manager can be set to different locales, without changing the database content. The database representation of a point in time is modeled as two integers, one for the starting time (0-2359) and one for the date (e.g. 20010303). If the locale is set to American standards times are presented in the "am/pm" format, and months are spelt out in English. If the locale is set to Swedish standards, times are presented in the military time format and months are consequently spelt out in Swedish.

The second way the system interacts with the user is through the messages that go together with a query template's SQL statement. This message string is language dependent, which means that it has to be completely translated if the natural language is changed.

6.2 DOMAIN RESULTS

In this section the domain-related results are described. The domain is identified and its representation is described (section 6.2.1 The

Domain and Its Representation) and the domain component's portability in general and its scalability in particular are accounted for (section 6.2.2 Portability and section 6.2.3 Scalability).

6.2.1 The Domain and Its Representation

Obviously, the domain definition and the sub-language definition are intimately connected. They can be viewed as the same thing viewed from two angles. The domain is an area of knowledge, and the sub-language is the collection of words and expressions used when communicating and reasoning about the domain. So, when describing the domain the distinction between the domain and its sub-language is sometimes elusive.

The concepts in the domain deal with the classification of programmes; occurrences and durations of programmes; content and credits information about programmes (such as synopsis, actors, year of production etc); and information about channels and their content and broadcasting details. This information is modeled as tables in a relational database (see section 5.3 Implementation). The tool used for reasoning about the database content is the formal query language SQL. Reasoning about the information is carried out by the domain knowledge manager. The database manager is a part of the domain knowledge manager module. Being general, the database manager can be reused. Domain-related database issues (i.e. SQL statements) are only present in the query template files, which are separate from – but used by – the database manager. This is essential for the portability/scalability issue discussed below.

As mentioned earlier, time and temporal information are important concepts in this domain. Objects in the television tableau domains are

identified with respect to their temporal characteristics. A large amount of work lies in dealing with temporal concepts.

In the corpus, there are dialogues that belong to the sub-language, but are not covered in the domain representation. One example of this is duration and ending times of television programmes. Another example is that the categories in the original information source (the Internet television site) do not contain as detailed information as certain user requests demand. There are shows that belong to the category FILM. If the user asks for action movies, he or she receives all movies, not just action movies. This is an information structure problem, which needs to be addressed in the future.

- Temporal concepts are crucial.
- The domain is represented in a relational database.
- The information structure is not optimal.

6.2.2 PORTABILITY

The MALIN architecture was reused to a large extent, even though the parser was the only module that was entirely kept. The original system architecture was developed using a modularized approach, which allowed for keeping the design, but changing some of the modules in the implementation. This section describes the current system architecture's portability.

The design resulted in a modularized system, in which the individual components can be exchanged and/or modified without having to rewrite the whole system. By refining individual modules, the system as a whole is improved.

The functionality of the chart parser module itself did not need to be changed. Obviously, when introducing a new domain the lexicon and grammar need to be rewritten to cover the new sub-language. Some rules could be directly translated from the CARS implementation (such as order numbering, e.g. "first", "second"), but most of the rules are domain and language dependent. Clear and complete documentation of the linguistic components (such as in section 5.2 Corpus Work) is essential when generalizing the grammar rules. Also, such documentation helps when reusing rules in new domains. To further increase the efficiency in developing grammars for new domains, generic methods are desirable.

The domain knowledge manager module was also designed and implemented from scratch. There are two things that need to be entirely rewritten in the domain knowledge manager when changing domain: the query template collection, and the mechanism for selecting the query templates. The knowledge base itself, the database, is where the actual domain change takes place. A domain change typically implies that new tables and relationships between tables occur. The database manager itself is not concerned with domain dependent SQL queries, but rather creates a connection to a database server and executes the SQL queries that are sent to it. Similarly, a *generic* temporal reasoner should be able to accommodate temporal expressions in any domain.

Moving on to the response manager, little has to be changed there as well. The principles, according to which the response manager formats the database output, are independent of the database content (e.g. names of tuple heads and format etc.). The database output accounts for one aspect of the system's output. The second aspect involves the message strings. These are intimately connected to the SQL statements and are thus placed in the query templates, which have to be rewritten anyway, as indicated above.

6.2.3 SCALABILITY

Scalibity refers to extending the existing television tableau domain. Scalability is to add new shows and channels to the database. By adding new shows, new titles, actors, starting times, descriptions etc. are also added. Scalability has not been tested in this iteration, but is a desirable feature in future versions. The current architecture is prepared for scalability, due to the distinction between the static and dynamic lexicon, and the distributed layout of the database tables. The distributed layout of the database tables follows good practise for non-redundant information, making database updates easier to manage.

The lexicon division makes it manageable to conserve the static parts of the linguistic representation of the domain (i.e. the sub-language), while changing the dynamic content.

To minimize the work effort associated with the domain aspect of the system, the shallow knowledge approach (2.2.3 Domain Coverage) proved to be useful and save development time.

7 Discussion

In this chapter the work of the design, and implementation of a dialogue system prototype is discussed. The purpose of this rapid prototyping approach was to establish what kind and amount of work that needs to be carried out in order to construct a dialogue system for the television programme tableau domain (i.e. in what direction work should be carried out, and where the main efforts should lie).

The empirical basis for the design and implementation is a dialogue collection – the project corpus – with the aid of a second corpus provided by the knowledge base content. Knowledge base content is detailed information about one week of television programmes represented in a relational database.

The design resulted in a modularized system, in which the individual components can be exchanged and/or modified without having to rewrite the whole system. By refining individual modules, the system as a whole can be improved.

7.1 THE LINGUISTIC ASPECT

In this section the linguistic results are discussed. First, suggestions how the sub-language and its representation can be improved are discussed (section 7.1.1 The Sub-Language and Its Representation). This discussion relates to how the robustness can be improved, which

is discussed in section 7.1.2 Robustness. Finally, the switch in natural language is discussed in section 7.1.3 Natural Language Extensibility.

7.1.1 The Sub-Language and Its Representation

The two views presented in section 6.1.1 The Sub-Language and Its Representation indicate that there is one static part and one dynamic part in the sub-language of television programmes. It is worth reminding the reader that the sub-language concerns simple task requests, and contains no system request vocabulary (see section 2.5.2 Reasoning About Domain Knowledge). Furthermore, the results indicate that special care needs to be taken to temporal phrases in this particular domain for two reasons: (a) temporal phrases are vaguely expressed (see section 2.5.3 Temporal Reasoning), and (b) temporal information is crucial in the domain and used to identify objects (see section 5.3.1 Knowledge Representation).

Most of the sub-language can be handled straight forward. Categories, channels, actors, presenters, directors and titles are precise in their linguistic use, and the linguistic representation mapping to the domain representation provides few problems. One problem is the notion of multi-word lexemes. Open nominal compounds have already been identified as presenting a problem (see section 2.4.4 Lexicon and Grammar), and since proper nouns, such as names on actors, directors and presenters, usually are multi-word lexemes, it is unfortunate that the chart parser has no support for this in the lexicon formalism. Proper nouns and titles constitute a major portion of the lexicon, and as of now, the coverage has to be accounted for by rules in the grammar. This calls for added functionality in the chart parser.

The problems presented by temporal phrases are more difficult to handle. In the television programme sub-language, temporal expressions are very

common – and they vary greatly in precision. The grammar responsible for time phrases can, like the temporal reasoner module, be troublesome.

The solution to have temporal phrases represented as both properties with temporal aspects, and as object-fields serves the purpose of portability. The temporal grammar only deals with properties (i.e. not object-fields), while the temporal reasoner ensures that the temporal representation is converted to the temporal object-fields. This approach ensures that the object-fields START and DATE always are compatible with the information in the database, while the properties information contains aspect/value representation of temporal expressions that is closer to the way users express them. It falls on the temporal reasoner module in the domain knowledge manager to resolve them. The double representation of temporal phrases means that the separate temporal grammar attributes to the increased portability of the system, since it can be used in a domain where temporal information need not be moved to object-fields.

In this thesis, the real end-users have been neglected on purpose. For the subsequent iterations it is advisable to introduce more user-centered work, such as gathering a large and varied project corpus using for example a WOZ[8] test. It is also worth pointing out that the users in such a test should be representative end users, and native speakers of the chosen natural language. The protoype developed in this project could be a useful tool for this sort of research.

The main disadvantage with the method used is that the use-cases are very limited, and no claims on the representativeness are put. The

[8] Wizard-of-Oz tests aims to study user interaction with a system that has not yet been implemented. The user is led to believe that he or she is interacting with a computer system, while in fact a human "wizard" is controlling the system in hide.

results of the work presented in this thesis are not generalizable, and merely form a starting point for a second iteration of the prototype. It is important to point out that the purpose of the first iteration is not to have a complete coverage – neither linguistically nor domain-wise.

The size of the corpus used for the construction of this prototype is not enough to account for all potential actions a user might take with a natural language dialogue system in the domain. If a bigger corpus was used, a wider range of coverage and a finer division between introduction, query and temporal phrases could be reached. The informal test to update the corpus only contained one subject, but even so provided useful additions to both lexicon and grammar. This implies that if more end-users were employed in the dialogue gathering, even more additions would be added. The closer the corpus translates in fidelity to the sub-language, the better the system's linguistic coverage will be.

The lack of generalizability and coverage in the corpus is not considered to be a significant problem, since the results (i.e. the "testament") are to be used directly as input values for the next design iteration when constructing the second version of this system (i.e. they are not used for generalization purposes).

To further enhance the linguistic capabilities of the dialogue system, a WOZ test could also contribute to meta-knowledge (system) question coverage. This obviously means an increase in the sub-language and implies a larger lexicon. Following the experience drawn from the modularization, yet another lexicon could be constructed containing information about the system and its capabilities.

Furthermore, certain information that is not really a part of the sub-language – but still anticipated – can be included. For example, it is possible to include a separate, static database (and accompanying lexicon)

with information about actors that users are likely to ask questions about (e.g. very famous actors), but that are not in any of the shows the current time period.

Extra-grammatical utterances are supported, due to the general rules. To allow for extra-grammatical utterances makes the system more linguistically robust. Looking ahead, handling extra-grammatical utterances might increase in importance when implementing spoken interaction.

For dialogue purposes, words such as "funny" could be included. Not because there is information in the database about "funny" shows, but because it might help the system assist the user finding shows that the user considers to be "funny". For example, the system could counter with "Well, what kind of shows do you think is funny?", or "Do you think Charlie Chaplin is funny?").

User-centered testing could also include response testing. The answer to the question "Is The Simpsons on BBC2 tomorrow?" could result in at least two types of answers and user testing should provide clues as to what kind of answer users in general want. Is a simple "Yes" or "No" sufficient, or do users want information about when Simpsons is actually screening. And if so, should the system include information about The Simpsons from other channels as well? (See section 2.6 Natural Language Generation).

Query phrases and temporal phrases are the obvious information-bearing components of a user utterance. The introduction phrases are more or less neglected, even if the grammar and lexicon are able to parse them. Taking care of the introduction phrases to identify level of courtesy and interaction style could be one way to account for personalized interfaces. This touches on social interfaces and is beyond the scope of this thesis.

7.1.2 ROBUSTNESS

Robustness in a dialogue system occurs in some form or another in all the system's modules. It is convenient to consider robustness on two levels:

- Utterance level
- Dialogue or discourse level

The utterance level robustness corresponds to the parser. In order to make a dialogue system robust on the utterance level the corpus coverage needs to represent the sub-language. Thus, the key to utterance level robustness improvement lies in the extensiveness and precision of the corpus. With a corpus of less magnitude, robustness can still be achieved using general grammar rules, as in the case of the system described in this thesis. The use of general rules to a certain extent can improve parser robustness. General rules allow the parser to skip words in the user utterance that is not included in the lexicon. Of course, the more general the rules are, more subtleties in the utterances will be missed. In combination with dialogue management, this feature becomes more useful since it provides robustness on both utterance and discourse level. To take robustness even one more step into the architecture, domain reasoning can also provide robustness.

The discourse (or dialogue) robustness corresponds to the dialogue manager, which has not been implemented in this version of the system. The dialogue manager in the MALIN architecture handles the OPMs in its native dialogue grammar format to build initiative/response nodes in a dialogue tree. This makes the system able to take initiative and ask the user for clarifying questions. Implementing a dialogue grammar would make the dialogue system more robust.

The robustness could be further increased concerning collaboration with the domain knowledge manager. With an advanced domain knowledge

manager, shortcomings in the corpus can be accounted for, and improve the robust behavior of the system as a whole. Temporal reasoning is very important in the television programme tableau domain (see section 6.1.1 The Sub-Language and Its Representation). Therefore, the status of the temporal reasoner module can be a troublesome bottleneck. It needs to be able to handle all temporal expressions that can possibly occur in the linguistic representation of temporal expressions. The temporal properties in the OPM (DAY, TIME and CLOCK) need to be accommodated by the temporal reasoner. For a generic, and more robust, temporal reasoner several other temporal properties should be added – for instance, MONTH and YEAR. Added functionality for error handling (such as identifying that "the 31st of February" as a non-valid date) is also desirable. A generic temporal reasoner is, at the time of writing, under development at NLPLAB.

7.1.3 NATURAL LANGUAGE EXTENSIBILITY

Modularization gives that parts of the existing system can be reused. It is desirable to try to isolate language dependent modules in order to make language transition smooth and manageable. In the televison tableau domain, the use of domain and language dependent query templates implies that the database manager and the response manager can be generic. In some cases, it is desirable that tuple headers and content are kept in a certain language, in some cases they need to be translated, and sometimes a combination is called for. In the case of TV shows, a direct translation of show titles (i.e. content) is often not desirable, while the tuple head itself (e.g. the word "title") needs to be translated.

To change natural language in the television tableau dialogue system, the following steps are required:

- Translate the Query template message strings.

- Set the locale for formatting time/date results in the response manager.
- Construct a language dependent lexicon, and a language dependent computational module.
- Construct a language dependent grammar to match the lexicon.
- Translate the database content (can either occur when deciding on information source, or translating tables, or in the response manager formatting system. For example, if the language switch is from English to Swedish, a rule in the response manager could be

```
if <incoming tuple header> = "Actor"
then set <tuple header> to "Skådespelare"
```

The advantage of the locale setting approach is of course that the response manager only requires a one-line change and the database remains untouched when switching natural language.

A WOZ setup to allow dialogue could serve as a tool for deciding whether the discourse is language dependent or not (i.e. are clarifications handled the same way in the Swedish discourse in the second-hand cars domain and in the English discourse of television programmes?).

There is an already-existing dialogue grammar developed for the CARS implementation of the MALIN architecture. It would be interesting to test whether that grammar is (a) domain dependent, and (b) language dependent, by implementing it in the English television tableau system.

To summarize the work effort for the linguistic aspect: The approaches and concepts described in chapter 2 Related Research, were used and worked well. Especially, the practical dialogue hypothesis (see section 2.2.2 Linguistic Coverage), partial parsing (described in section 2.4.3 Chart Parsing), computational lexicon approach (see section 2.4.4 Lexicon and Grammar), and robust grammar rules (see section 2.4.3 Chart Parsing,

and section 4.2 The Chart Parser Flexchart) proved to minimize the workload.

7.2 THE DOMAIN ASPECT

This section discusses the results presented in section 6.2 Domain Results. First, the domain results are discussed in section 7.2.1 The Domain and Its Representation. Second, portability is discussed (section 7.2.2 Portability). This section is concluded with a discussion about scalability (section 7.2.3 Scalability).

7.2.1 THE DOMAIN AND ITS REPRESENTATION

Figure 28 shows how the domain, its representation, the sub-language and its representation are connected to each other through a reasoning agent (either a human being or a dialogue system).

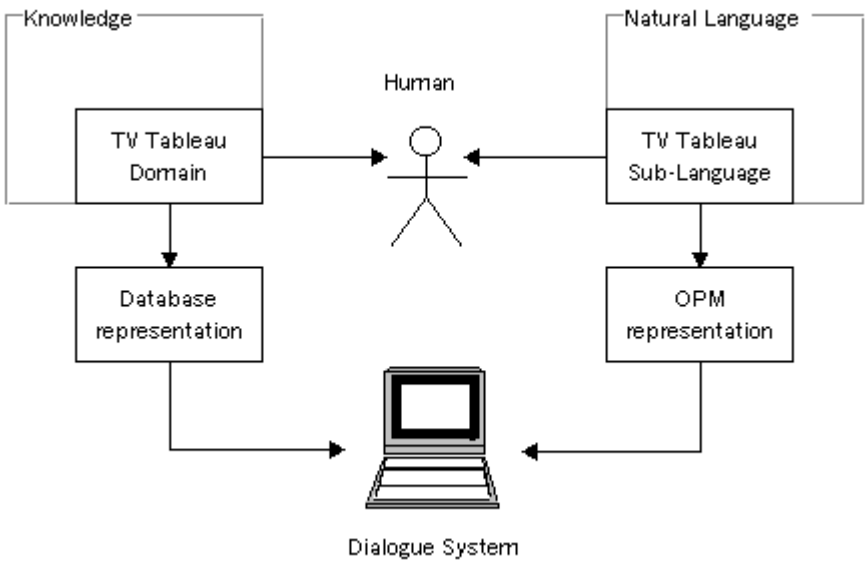

Figure 28 The connection between domain representation, sub-language representation and dialogue system.

The dialogue system in Figure 28 shows the ideal dialogue system (i.e. a dialogue system with optimal domain representation, optimal sub-

language representation and an ideal domain knowledge reasoner module can exchange a human reasoner for the particular domain). This is obviously a faulty view of both human beings and dialogue systems, but it serves the purpose of clarifying how representations of domain and sub-language are connected to the dialogue system. It is worth noting that the human mental representations of language and knowledge are left out in the figure. Dealing with mental representations of language and knowledge is not the topic of this thesis.

It is desirable that the domain knowledge manager is as generic as possible. In order to make a module generic, its sub-parts (i.e. the temporal reasoning module and the database manager) should be developed to be more general. The implemented temporal reasoner handles the utterances – and variations – present in the project corpus. Its interface is directly dependent on the OPM knowledge representation, but this can be altered.

Since temporal information is crucial in the domain, this is modeled as primary keys in the television programme objects. START and DATE, along with CHANNEL, constitutes the key fields in the OPM.

There are two problems with representing the domain information in a relational database. First, there is a gap between the sub-language and the information in the database. Sometimes the user might ask about information not included in the database. For example, the utterances "action movie" and "drama" both denote the category "film" in database). This can be accounted for in at least two ways:

- The finer-grained user classification can be ignored. This is the current status of the system. General rules skip the word

"action" in the phrase "action movies" and delivers *all* movies. The user has to find out what kind of movie each one is by further questioning. This approach is not optimal, since the user does not get an explanation of what went wrong.

- A better way is to utilize meta-knowledge in the domain knowledge manager to report on false presuppositions (see section 2.6 Natural Language Generation). This requires a functioning system model, allowing the domain knowledge manager to explain how for example movies are categorized. The response could in this case be something like: "There is no sub-classification of movies. Retrieving all movies: …"

There is a similar gap that is dependent on the representation. The original XML source does not contain *explicit* information about duration and stopping time. However, since each channel can only broadcast one programme at a certain time on a certain date, stopping time (and duration) can be derived by finding the next programme on the same channel. When the XML documents are loaded into the database (of the current layout) this information is lost. An improvement in the domain knowledge manager could be the ability to infer duration and stopping times, even if this information is not expressed in the database. If we know the starting time of every show, the stopping time (and thus show duration) can be derived from looking at the closest next or previous show starting time on the given channel.

Information structure in the database can pose another problem, even if we have covered it in the sub-language. Sometimes actor and director information is included in DESCRIPTION, but skipped in the CREDITS table. For example: the query "Show me the synopsis for The

Yakuza" can give actor information if for example the main actor is listed in the synopsis. Often, this implies that the main actor in the CREDITS table is missing. Thus, the query "Who is the main actor in The Yakuza" gives no information. This problem is due to the quality of the original information source (i.e. the chosen television site on the Internet).

These two information structure problems imply that work with content-centered corpus, i.e. parsing the XML documents before adding it to the database, should be carried out.

Finally, a note on shortcomings in SQL is in place: The more specified an SQL statement is, the longer it takes for the MySQL client to process the query. This is due to limitations in the SQL language, which is unfortunate for the users of the television system. It is, so to speak, easier for the system to quickly find and print all movies on all channels at all times of a particular day, than it is to find out if a certain movie on a specific channel has a given actor or actress. The more specific a user is in his or her query (i.e. the more pieces of information he or she provides), the more complex the SQL statement usually becomes. A complex SQL statement takes longer to process than a simpler one, rendering the waiting time longer for a specific request. This results in that the more specific information a user provides, the longer he or she will have to wait for the response.

This phenomenon should be placed in perspective: when a very specific user-request has been processed and presented, it is hopefully exactly the right information the user wanted; whereas a general query usually results in follow-up questions, whose combined process time might exceed the specific query's process time. It remains to be verified if this is the case or not. More importantly, it remains to be

verified whether the user *experiences* this to be the case or not. Since the tables in the database are normalized (see section 2.5.1 Relational Databases), the problem is not due to the database structure. The implication of this is that an SQL database might not be the optimal representation for this domain.

7.2.2 PORTABILITY

As a reminder, extensibility is a general characteristic that spans the change of natural language, portability, scalability and transportability. In this section, the system's portability is discussed.

As mentioned in section 7.2.1 The Domain and Its Representation, temporal information is first processed by a separate temporal grammar, which yields in a properties representation. The two temporal object-fields (START and DATE) are updated by the temporal reasoner, using the information in the temporal properties. As long as the OPM domain representation is used, the temporal grammar can thus be reused – even in domain representations where the temporal aspects are not included as object-fields. The temporal reasoner needs to accommodate this in other domains. This supports the notion of modularization, since even the grammar (which is a sub-module to the parser, which is a sub-module to the linguistic component, etc.) is further modularized to increase portability.

The SQL statements are domain dependent. With the statements (from the query templates in this implementation) separated from the database manager, the effort of changing domain is minimized. Similarly, the response manager can be generalized, since it is disconnected from the query templates. This enhances not only the

portability of the system as a whole, but helps in other extensibility issues, such as natural language change.

7.2.3 SCALABILITY

Scalability is a specific case of portability, namely the expansion of the existing domain. This has not been done in the prototype described herein, but it is desirable that this feature is included in following versions.

Scalability can be accounted for from two directions – or angles: Weak scalability (updating the database content, and extending the dynamic lexicon in the parser) and true scalability (rebuilding the database with types of information about television programmes previously not recorded). If the database setup is changed, the domain knowledge reasoner and the database have to be reconstructed, while the lexicon and grammar only needs extension. One might argue that there in this case is a portability issue connected to the domain knowledge reasoner module and the database, while there is a scalability issue connected to the interpretation module. True scalability has not been dealt with, but might be a desirable feature in future implementations. The weak scalablity (just "scalability" henceforth) of this system deals with updating the database content, and updating the lexicon, while the changes in the interpretation module is limited to updating the dynamic contents of the lexicon.

In this prototype the domain is static. That is, the database content represents a "snapshot" of one week of television programmes. In a future version of the television query system the domain is to be constantly upscaled, and automatically updated. This requires a new, more complex architecture, which is suggested in chapter 8.2.2 Suggestions for a New Architecture.

7.3 IMPLEMENTATION ISSUES

This section discusses the transportability (section 7.3.1 Transportability) and modularization (section 7.3.2 Modularization) issues, from the implementation of the dialogue system prototype.

7.3.1 TRANSPORTABILITY

Transportability refers to the changes in computational platform, hardware and programming language(s). In this thesis the platform change is from Unix to Linux. Linux is the operating system of the Nokia Mediaterminal, and Unix is the platform used by NLPLAB when the parser and the MALIN architecture were developed. Since these platforms are similar, few problems were encountered. Unix and Linux are similar enough to provide few transportability difficulties. The only significant problem was the Lisp-based chart parser.

The chart parser provided by NLPLAB is the only module that was entirely kept. The only significant problem concerned the fact that the parser is written in Lisp, and requires a separate, proprietary Lisp compiler that does not abide to the desirable open systems standards. Flexchart is not very transportable (see section 4.2 The Chart Parser Flexchart). To come to terms with the transportability problem of the parser I suggest that the parser be rewritten in Java. There are five reasons I recommend rewriting the chart parser.

First, since the current Chart parser Flexchart is written in Lisp, it requires an expensive compiler, whereas the Java programming environment is free. Second, the rest of the system framework is Java-based, and the handling of objects and strings would be faster if the whole system would be Java-based. Third, future knowledge engineers and system builders would not be required to be familiar with Lisp –

Java knowledge would be sufficient. Fourth, Java is platform-independent, and a system written completely in Java would be transportable to new operating systems. Fifth, the Lisp-based parser and the Java-based dialogue manager communicate via a socket. Related to the discussion about robustness is the overall system robustness. The programming language aspect is connected to this. The connection between the Lisp-based parser and the Java-based dialogue manager module is via a socket stream, which can fail. Sockets are not fail-safe and this implies a weak link in the overall robustness of the system. By employing a Java-based parser that particular route to potential failure is eliminated.

7.3.2 MODULARIZATION

Modularization makes it possible to divide the system into manageable pieces, and high modularization is generally agreed upon (see section 2.2.4 System Architecure and Modularization). It seems like modularization can be stretched far. It has already been suggested that it is advantageous to modularize the linguistic, as well as the domain, components. For example, the linguistic component can consist of one interpreter, one response generator and one dialogue management module. The television tableau dialogue system prototype implementation shows that even these modules can be made more extensible and easier to implement if they are modularized. For example, the television tableau dialogue system prototype has two grammar modules, one static lexicon and one dynamic lexicon.

High modularization requires a good framework for plug-ins, and good interface documentation is essential. Documentation of the MALIN system is currently sparse and insufficient, rendering it difficult for a new user to quickly grasp the system.

I suggest the system should abide to open systems standards, in order to minimize acquisition cost and efforts, by taking advantage of reuse. This includes rewriting the chart parser in Java (see section 7.3.1 Transportability).

The workload effort benefits from taking advantage of already existing modules (see section 2.2.4 System Architecure and Modularization, and chapter 4. The MALIN Framework). By using the modular approach, development time was saved.

8 Conclusion and Future Research

In this thesis the work of designing and implementing a dialogue system prototype has been described and discussed. The main results presented in chapter 6 Results, are discussed in chapter 7 Discussion, and lead to the conclusions presented in this chapter. This chapter is concluded with some suggestions for future research. Future research includes work on a second version of the prototype – taking it one step closer to a robust dialogue system suitable for the future e-home.

8.1 CONCLUSIONS

It was found that having one static lexicon and one disposable dynamic lexicon enhances scalability. Furthermore, dividing the grammar into sub-modules, such as a separate temporal grammar, enhances both scalability and portability. A language dependent, but domain *independent*, temporal grammar that can be used in new sub-languages, which are represented in OPMs enhances portability. It also makes scalability increase, since extending the domain might include extending some parts of the sub-language, which can be modelled in separate grammars to reflect this extension.

Temporal representation in both properties and object-fields yields in a portable system, since the temporal grammar only deals with

properties, and is suitable in a domain where temporal information is not used as object-fields. The temporal reasoner in the domain knowledge manager module is responsible for updating the temporal object-fields, which are crucial for the linguistic representation in domains where temporal information is used as object-fields (such as the television tableau domain described in this thesis).

It is my belief that modularization can be taken further than the MALIN architecture suggests. One example is the temporal grammar and keeping temporal properties separated from object-fields.

Linguistic robustness is divided into utterance level robustness and discourse level robustness. Employing a larger corpus, in combination with general rules, enhances utterance level robustness. A larger corpus can be gathered with a WOZ test, or by logging the interaction with the prototype described in this thesis. Employing a dialogue grammar, and allowing it to collaborate with an advanced dialogue knowledge manager can enhance discourse level robustness.

The existing chart parser does not support lexicons with multiword lexemes, such as names and titles, which is unfortunate in this particular domain. Generally, this feature is desirable as well, since open nominal compounds are used frequently in English.

There are two "gaps" in the information source that needs to be bridged. The first gap is the lack of information in the original source. This includes for example fine-grained categorization of television programmes. The second gap relates to the lack of structure in the original source. This implies that more work when parsing the XML documents, before loading the database is needed. Also, a more advanced domain knowledge reasoning mechanism is desirable.

8.2 FUTURE RESEARCH

Since the field of dialogue system construction is fairly new, a lot of things need to be done generally. This section first describes some rough directions where further research could be carried out. Secondly, some suggestions about a second iteration of the television tableau dialogue system prototype are presented.

8.2.1 GENERAL ISSUES

In general, theory is needed in the field of language technology and dialogue system design. The terminology needs to be set, since there is a confusing inconsistency in the literature.

Furthermore, methodology on different levels is lacking. The method used in this thesis, verifies the usability of Degerstedt and Jönsson's method (see chapter 5 The TV-Programme Dialogue System). Rapid prototyping in general is not unheard of (e.g. Extreme programming, and even language technology-specific methods such as CSLU), but a complete method for the entire system building chain of stages is desirable. On the other hand, methodology on a lower level is sought after. I believe that for example grammar development can be more efficient, and allowing for systematic reuse of old grammars, with an adequate method. This is lacking today, and calls for future research.

The intention with a dialogue system in the television tableau domain is to extend the interaction to spoken or multi-modal interaction, which incorporates speech technology.

True scalability issues needs to be tested in this domain. What happens if the database is configured in a different way, but in the same domain? We have seen that weak scalability is handled, and I have suggested that the interpreter and the domain knowledge

component handle true scalability differently. If this is the case remains to be verified.

The waiting times on SQL queries should be tested to answer question if user's generally prefer quick-and-dirty approach with sloppy expressions and lots of data and gradually refining it, or if they prefer a carerfully utterance and wait for a high-quality response. The user's *experience* is the crucial aspect. The user's feeling of efficiency with the dialogue system in contrast to conventional searching in the newspaper or on the Internet is important to identify.

8.2.2 Suggestions for a New Architecture

Two main improvements are introduction of a dialogue grammar, and the addition of dynamic knowledge base updates. Furthermore, there are a multitude of smaller improvements suggested above. The architecture in Figure 29 shows the layout of such a design.

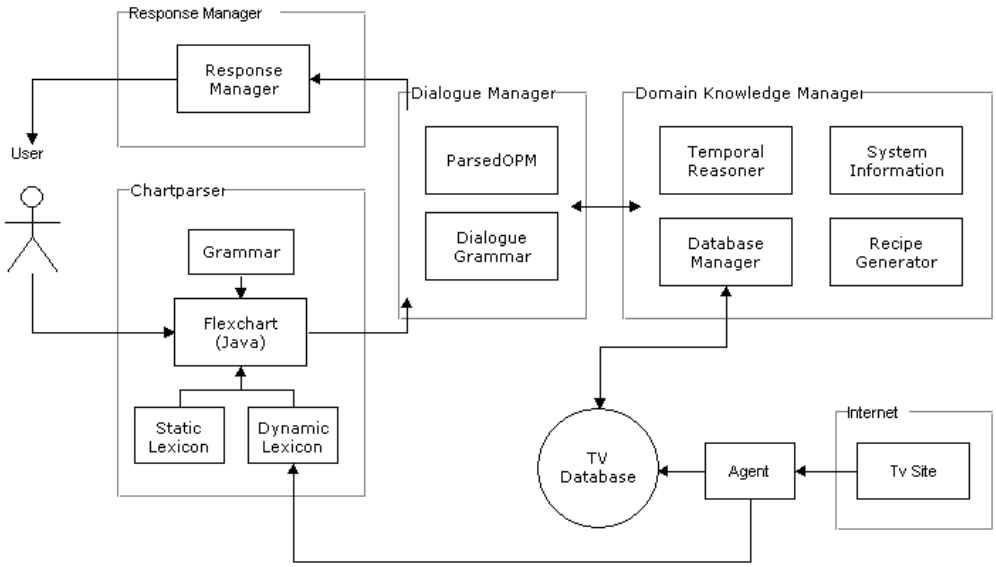

Figure 29 A new system architecture for the television tableau domain.

The main ideas of a modularized approach are conserved. The design and function of the parser is kept, but as mentioned (see section 7.3.1 Transportability) I suggest the parser to be implemented in Java.

A dialogue manager should be incorporated in the system, to raise the system's complexity to at least level 2 in Table 1 in section 2.2 Dialogue Systems. To include a dialogue manager allows for utilizing markers in the OPM, which implies adding functionality to the grammar and the lexicon. A dialogue grammar should be added to the dialogue manager. The dialogue grammar can be viewed as a context-free grammar with categories and rules. The purpose of the grammar is to construct a dialogue tree and allow for focus handling.

More application-specific research includes testing of the portability of the dialogue grammar developed for the CARS application (see section 4.1 The MALIN System Architecture). The testing can also span language dependency of the dialogue grammar. There is no evidence to the author's knowledge that discourse in the same domain but in a different language is generalizable, and this remains to be tested.

In order to make the domain knowledge manager module more functional, it is updated with a system information database, a generic temporal reasoner, and a recipe generator (Flycht-Eriksson, 2000).

One interesting part is to have a software agent, or module, retrieving television tableaus from television sites on the Internet, and loading the content in the relational database on for example a weekly basis. This procedure was carried out manually in the first version. Information represented in XML was taken from an Internet site and converted to text files, which were loaded in the relational database. An agent could carry this procedure out automatically, providing a dynamically updated knowledge base. Upscaling the database is not enough. The lexicon needs to be updated with titles, actors, presenters and directors. This falls under the agent's responsibilities as well. Related to the lexicon extension is the housekeeping of both database and lexicon. Since the tableaus for one week of television programmes consist of some 4700+ entries, a constantly growing lexicon and database soon get cumbersome. It is desirable that old material is discarded on a regular basis. One solution for this problem is to keep the static and dynamic lexicon separated, and simply discard the old dynamic lexicon and replace it with a new collection of titles, actors etc. every update. The agent (or module) needs to be flexible, since it is dealing with information in a dynamic environment. This fact puts high requirements on the agent's abilities.

In order to further enhance the system's capabilities, a *system model* ("system information" in Figure 29) should be provided in the domain knowledge manager. Ideally, this provides the system with information about what the system's capabilities are. The system would, so to speak, know what it knows. Furthermore, the system can use this to along with a *user model* (not present in Figure 29). The system would then know that the user knows what the system knows.

This would make the system address unexpected user utterances with greater confidence and try to match them with the user and system models. This is certainly an interesting area for future research.

9 References

Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A. (1998). An Architecture for a Generic Dialogue Shell. In *Natural Language Engineering, 1*, 1-15.

Allen, J., Ferguson, G., Stent, A. (2001a). An Architecture For More Realistic Conversational Systems. In *Proceedings of IUI '01*, 14-17. Santa Fe, New Mexico, USA.

Allen, J.F., Byron, D.K., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A. (2001b). Towards Conversational Human-Computer Interaction. In *AI Magazine, 2001*, 1-9.

Amsler, R. (1989). Research Toward the Development of a Lexical Knowledge Base for Natural Language Processing. *Proceedings of the twelfth annual international ACMSIGIR conference on Research and development in information retrieval*, 242-249. Cambridge, MA, USA.

Androutsopoulos, I., Ritchie, G.D., Thanisch, P. (1994). Natural Language Interfaces to Databases – An Introduction. In *Journal of Natural Language Engineering*, Mars 1995. Cambridge, England: Cambridge University Press.

Bell, J.E., Rowe, L.A. (1992). An Exploratory Study of Ad Hoc Query Languages to Databases. In *Proceedings of the 8th International Conference on*

Data Engineering, Tempe, Az. (p. 606-613). IEEE Computer Society Press, February 1992.

Clark, H.H., (1996). *Using Language*. Cambridge, England: Cambridge University Press.

Cowie, J., Lehnert, W. (1996). Information Extraction. In *Communications of the ACM*, *Vol. 39, No 1*, 80-91.

Dahlbäck, N., Jönsson, A. (1999). Knowledge Sources in Spoken Dialogue Systems. In *Proceedings of Eurospeech '99*. 1523-1526. Budapest, Hungary.

Degerstedt, L., Jönsson, A. (2001). A Method for Iterative Implementation of Dialogue Management.In *2nd IJCAI Workshop on Knowledge And Reasoning In Practical Dialogue Systems*. Seattle: August 2001.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, *6*, 451-455.

Flycht-Eriksson, A. (2000). A Domain Knowledge Manager for Dialogue Systems. In *Proceedings of ECAI*. Berlin: 2000.

Flycht-Eriksson, A. (2001). *Domain Knowledge Management in Information-providing Dialogue Systems*. (Linköping Studies in Science and Technology, dissertation No.890). Linköping: Department of Computer and Information Science, Linköping University.

Flycht-Eriksson, A., Jönsson, A. (2000). Dialogue and Domain Knowledge Management in Dialogue Systems. In *Proceedings of the 1st SigDial Workshop*. Hong kong: 2000.

Hafner, C.D., Godden, K. (1985). Portability of Syntax and Semantics in Datalog. In *ACM Transactions on Office Information Systems*, *Vol. 3, No. 2*, 141-164.

Hansen, M. (1998). *En robust, flexibel och itegrerbar chart parser*. Thesis Work. Linköping: Department of Computer and Information Science.

Jacobs, P.S., Rau, L.F. (1988). *Natural Language Techniques for Intelligent Information Retrieval*. In SIGIR 1988, 85-99.

Janssen, T. M. (1997). Compositionality. In van Benthem, J. and ter Meulen, A. (Eds.), *Handbook of Logic and Language*, chap. 7, pp. 417-473, North-Holland, Amsterdam.

Jönsson, A. (1993). *Dialogue Management for Natural Language Interfaces*. (Linköping Studies in Science and Technology, dissertation No.312). Linköping: Department of Computer and Information Science, Linköping University.

Jönsson, A. (1997). A Model for Habitable and Efficient Dialogue Management for Natural Language Interaction, In *Natural Language Engineering, 3 (2/3)*, 103-122. Cambridge, England: Cambridge University Press, 1997.

Jönsson, A., Strömbäck, L. (1998). Robust Interaction through Partial Interpretation and Dialogue Management. In *Proceedings of Coling-ACL '98*, Montreal, Canada.

Jurafsky, D., Martin, J.H. (2000). *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. New Jersey: Prentice Hall.

Kelley, J. F., (1983). An Empirical Methodology for Writing User-Friendly Natural Language Computer Applications, *Proceedings of the CHI'83*, pp. 193-196.

McRoy, S., Ali, S.S., Restificar, A., Channarukul, S. (1999). Building Intelligent Dialog Systems. In *Intelligence*, *Spring 1999*, 14-23.

Merkel, M. (1988). A Novel Analysis of Temporal Frame-Adverbials. In Coling, Budapest: 1988.

Norman, D. (1988). *The Design of Everyday Things*. New York: Doubleday.

O'Shea, T., Eisenstadt, M. (1984). Artificial Intelligence. Tools, Techniques, and Applications. Harper & Row Publishers.

Russel, S., Norvig, P. (1995). *Artificial Intelligence. A Modern Approach*. New Jersey: Prentice Hall.

Santamarta, L. (2001). Natural Language Generation. Lecture in Language Technology, University of Linköping.

Smith (1981). Semantic and Syntactic Constraints on Temporal Interpretation. In Tedeshi, P.J. & Zaenen, A (eds.), *Syntax and Semantics, Vol. 14*, 213-238.

Strömbäck, L. (1996). *User-Defined Constructions in Unification-Based Formalisms*. (Linköping Studies in Science and Technology, dissertation No.461). Linköping: Department of Computer and Information Science, Linköping University.

Thompson, B.H., Thompson, F.B., (1985). ASK is Transportable in Half a Dozen Ways. In *ACM Transaction of Office Information Systems*, *Vol. 3, No. 2*, 185-203.

Watt, W. C., (1968). Habitability, *American Documentation*. July, pp 338-351.

Wiren, M. (1992). *Studies in Incremental Natural-Language Analysis*. (Linköping Studies in Science and Technology, dissertation No.292). Linköping: Department of Computer and Information Science, Linköping University

Appendix A: Glossary

chart parsing	Parsing algorithm developed by Early (1970) that consists of a single left-to-right pass that fills an array (the chart). For each word position in the sentence, the chart contains a list of states represented the partial parse generated so far.
clock calendar phrase	temporal phrases such as "midnight", "at noon" etc.
computational lexicon	Lexicon consisting of word stems with morphological rules for derivating words. A computational lexicon reduces computation and development time. Also known as a mini-lexicon.
context-free grammar	A grammar consisting of a set of rules for predefined categories. Each rule expresses how categories can be grouped together. Also known as a phrase-structure grammar.
corpus	A collection of utterances in a sub-language used in a certain domain.
customization	see *Extensibility*
DAG	see *Directed Acyclic Graph*
deictic phrase	temporal phrase anchored in time of speech, such as "last ... " and "... "

	as "last week" and "tomorrow".
dependent phrase	temporal phrase anchored in another given time in the context, such as "after" and "later".
dialogue	A discourse between two entities, where turn-taking and grounding occurs.
directed acyclic graph	(DAG). The graphical representation of a feature structure.
domain	An defined area of knowledge, for which a system can be built.
extensibility	A dialogue systems ability to change natural language, domain, and computational platform. Also known as customization.
feature structure	An attribute-value structure that can be used for information representation. Can be represented as a matrix, a graph, and an equation.
Flexchart	A Lisp-based chart parser provided by NLPLAB.
foreign key	An artificially constructed key for identifying tuples in a relational database.
formal language	An artificial language, that is rigidly defined, such as Java, SQL and Lisp.
genre	A type of discourse with specific features and content.
habitable	The user never feels restrained by prohibited sentences in the interaction with a natural language interface.
lexicon	A word list containing the words used in a sub-

	language.
LINLIN	A generic dialogue system framework developed at NLPLAB.
MALIN	A refined dialogue system framework derived from the Linlin architecture, consisting of extended domain knowledge reasoning capabilities.
marker	see *OPM*
meta-knowledge question	A user question about the capabilities of the dialogue system (such as "Can I use slang when I interact with you?").
mini-lexicon	see *Computational Lexicon*
modal question	Questions about the relation between information in the database.
natural language	Human languages, such as Swedish or English, which can be written and/or spoken.
natural language generation	When a speaker uses knowledge about a language to decide what to say.
NHC	Nokia Home Communications
NL extensibility	Natural language extensibility. The change of interacting natural language in a dialogue system.
NLG	see *Natural Language Generation*
NLPLAB	The Natural Language Processing Laboratory
object	see *OPM*
open nominal	Expressions such as "action movie" or "ice

compound	cream".
open system standards	A standard for system components that allows for different vendors to build on existing modules to save development cost and time and handle evolving needs.
OPM	A unifiable unification structure consisting of objects and properties. Used for dialogue management and domain knowledge reasoning.
parse tree	A graphical representation of a sentence parsed with a context-free grammar.
parsing	The process of assigning part of speech attributes to each word in a sentence.
portability	A sub-case to extensibility meaning how well a dialogue system architecture can handle a domain change.
practical dialogue	A discourse focused on accomplishing a well-defined task, such as a task-oriented dialogue.
primary key	An artificially constructed key for identifying tuples in a relational database.
property	see *OPM*
query block	The smallest meaning-bearing phrase within a query phrase.
query phrase	One of three phrase components identified in the television programme domain corpus. See also *Introduction Phrase* and *Temporal Phrase*.
scalability	A sub-case of portability meaning how well a

	dialogue system architecture can handle domain upscaling.
sub-language	A set of a natural language that is used within a genre.
system question	see *meta-knowledge question*
tokenization	The process of identifying meaning-bearing lexemes in a corpus.
transportability	A sub-case of extensibility, referring to how well a dialogue system architecture handles a change in computational platform and/or programming language.
unification	The process of comparing two feature structures, and returning one feature structure that represent the union of the two original structures.
utterance	A generic term referring to any mode of communication.
WOZ	Wizard-of-Oz study. Method for corpus gathering

Appendix B: SQL State-ments

The following SQL statements can be generated, depending on what information the user has given.

- **<x>** means "contents of variable x" (e.g. "< 1800").
- **TIME** means that START and/or DATE exists.
- **EMPTY** means that that property only has aspect, but no (or generic) argument.

1 STATEMENT TEMPLATE:

user has given TIME

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, programme
WHERE title.ForeignKey = programme.PrimaryKey
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

What is on TV right now?

2 STATEMENT TEMPLATE:

user has given TIME and CATEGORY.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
```

```
FROM title, programme, category
WHERE title.ForeignKey = programme.PrimaryKey
AND category.ForeignKey = programme.PrimaryKey
AND category.category = <category>
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

Are there any movies tonight?

3 STATEMENT TEMPLATE:

user has given TIME and TITLE.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel, programme.start
FROM title, programme
WHERE title.ForeignKey = programme.PrimaryKey
AND title LIKE <title>
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

Is Star Wars on this weekend?

4 STATEMENT TEMPLATE:

user has given TIME, ACTOR/DIRECTOR/PRESENTER.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, programme, credits
WHERE title.ForeignKey = programme.PrimaryKey
AND credits.ForeignKey = programme.PrimaryKey
AND credits.creditsAspect LIKE <creditsValue>
ORDER BY programme.date, programme.start
```

Is Shannon Tweed on tv tomorrow night?

5 STATEMENT TEMPLATE:

user has given TIME, CATEGORY and ACTOR/DIRECTOR/PRESENTER.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, programme, category, credits
WHERE title.ForeignKey = programme.PrimaryKey
AND title.ForeignKey = category.ForeignKey
AND category.category = <category>
```

```
AND credits.creditsAspect LIKE <creditsValue>
AND credits.ForeignKey = title.ForeignKey
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

List all movies directed by Sydney Pollack today?

6 STATEMENT TEMPLATE:

user has given TITLE and empty ACTOR/DIRECTOR/PRESENTER.

user expects TITLE and ACTOR/DIRECTOR/PRESENTER.

```
SELECT title.title, credits. creditsAspect
FROM title, credits
WHERE title.ForeignKey = credits.ForeignKey
AND title.title LIKE <title>
```

Who directed Omega Doom?

7 STATEMENT TEMPLATE:

user has given TITLE.

user expects TITLE, CATEGORY and DESCRIPTION.

```
SELECT title.title, category.category,
description.description
FROM title, description, category
WHERE title.title LIKE <title>
AND description.ForeignKey = title.ForeignKey
AND category.ForeignKey = title.ForeignKey
```

What is Omega Doom about?

8 STATEMENT TEMPLATE:

user has given CATEGORY.

user expects CHANNEL and TITLE

```
SELECT DISTINCT programme.channel, title.title
FROM programme, category, title
WHERE category.ForeignKey = programme.PrimaryKey
AND title.ForeignKey = category.ForeignKey
AND category.category = <category>
ORDER BY programme.channel
```

Is there a news channel?

9 STATEMENT TEMPLATE:

user has given CHANNEL and TIME.

user expects TITLE, START and DATE.

```
SELECT title.title, programme.start
FROM title, programme
WHERE title.ForeignKey = programme.PrimaryKey
AND programme.channel = <channel>
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

What is on BBC2 before seven pm on Tuesday?

10 STATEMENT TEMPLATE:

user has given CHANNEL, TIME and CATEGORY.

user expects TITLE, START and DATE.

```
SELECT title.title, programme.start, programme.date
FROM title, programme, category
WHERE title.ForeignKey = programme.PrimaryKey
AND category.ForeignKey = title.ForeignKey
AND programme.channel = <channel>
AND category.category = <category>
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

Are there any movies on BBC1 this weekend?

11 STATEMENT TEMPLATE:

user has given CHANNEL, TIME, CATEGORY and ACTOR/DIRECTOR/PRESENTER.

user expects TITLE, START and DATE.

```
SELECT title.title, programme.start, programme.date
FROM title, programme, category, credits
WHERE title.ForeignKey = credits.ForeignKey
AND category.ForeignKey = programme.PrimaryKey
AND credits. ForeignKey = programme.PrimaryKey
AND programme.channel = <channel>
AND programme.start = <start>
AND programme.date = <date>
AND credits.creditsAspect LIKE <creditsValue>
AND category.category = <category>
ORDER BY programme.date, programme.start
```

Are there any movies directed by Sydney Pollack on Channel5 tomorrow night?

12 STATEMENT TEMPLATE:

user has given CHANNEL, TIME, CATEGORY and empty ACTOR/DIRECTOR/PRESENTER.

user expects TITLE and ACTOR/DIRECTOR/PRESENTER.

```
SELECT title.title, credits.<creditsAspect>
FROM title, programme, category, credits
WHERE title.ForeignKey = programme.PrimaryKey
AND category.ForeignKey = programme.PrimaryKey
AND category.category = <category>
AND programme.channel = <channel>
AND credits.ForeignKey = title.ForeignKey
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

Who is the main actor in tonights action movie on BBC1?

13 STATEMENT TEMPLATE:

user has given CATEGORY, ACTOR/DIRECTOR/PRESENTER.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, programme, category, credits
WHERE title.ForeignKey = credits.ForeignKey
AND category = <category>
AND credits.creditsAspect LIKE <creditsValue>
AND category.ForeignKey = title.ForeignKey
AND credits.ForeignKey = programme.PrimaryKey
ORDER BY programme.date, programme.start
```

What movies are directed by Sydney Pollack?

14 STATEMENT TEMPLATE:

user has given empty TIME and TITLE.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, programme
WHERE title.ForeignKey = programme.PrimaryKey
AND title.title LIKE <title>
ORDER BY programme.date, programme.start
```

When is The Simpsons?

15 STATEMENT TEMPLATE:

user has given empty TIME and ACTOR/DIRECTOR/PRESENTER.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, programme, credits
WHERE credits.ForeignKey = programme.PrimaryKey
AND title.ForeignKey = credits.ForeignKey
AND credits.creditsAspect LIKE <creditsValue>
ORDER BY programme.date, programme.start
```

When is Shannon Tweed on tv?

16 STATEMENT TEMPLATE:

user has given CHANNEL, empty TIME and ACTOR/DIRECTOR/PRESENTER.

user expects TITLE, CHANNEL, START and DATE.

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, credits, programme
WHERE title.ForeignKey = credits.ForeignKey
AND credits.ForeignKey = programme.PrimaryKey
AND programme.channel =  channel +
AND credits. creditsAspect LIKE <creditsValue>
ORDER BY programme.date, programme.start
```

When is Rutger Hauer on Channel5?

17 STATEMENT TEMPLATE:

user has given CHANNEL, TIME and TITLE.

user expects START and DATE.

```
SELECT programme.start, programme.date
FROM programme, title
WHERE programme.channel = <channel>
AND title.title LIKE <title>
AND title.ForeignKey = programme.PrimaryKey
AND start = <start>
AND date = <date>
ORDER BY programme.date, programme.start
```

Is The Simpsons on BBC2 this week?

18 STATEMENT TEMPLATE:

user has given CATEGORY and TITLE.

user expects TITLE and CATEGORY (as Y/N).

```
SELECT title.title, category.category
FROM title, category
WHERE title LIKE <title>
AND title.ForeignKey = category.ForeignKey
```

Is Superbowl a movie?

19 STATEMENT TEMPLATE:

user has given CHANNEL and CATEGORY.

user expects TITLE, START and DATE.

```
SELECT title.title, programme.start, programme.date
FROM title, programme, category
WHERE programme.channel = <channel>
AND category = <category>
AND category.ForeignKey = programme.primaryKey
AND title.ForeignKey = category.ForeignKey
ORDER BY programme.date, programme.start
```

What sport events are featured on BBC2?

20 STATEMENT TEMPLATE:

user has given TITLE and empty DESRIPTION.

user expects TITLE, CATEGORY and DESCRIPTION.

```
SELECT title.title, category.category
description.description
FROM title, description, category
WHERE title.ForeignKey = description.ForeignKey
AND category.ForeignKey = title.ForeignKey
AND title LIKE <title>
```

What is Electra about?

21 STATEMENT TEMPLATE

user has given TIME, empty ACTOR/DIRECTOR/PRESENTER.

user expects TITLE, ACTOR/DIRECTOR/PRESENTER, CHANNEL, START and DATE.

```
SELECT title.title, credits.creditsAspect,
programme.channel, programme.start, programme.date
FROM title, programme, credits
WHERE title.ForeignKey = programme.PrimaryKey
```

```
AND credits.ForeignKey = programme.PrimaryKey
AND programme.start = <start>
AND programme.date = <date>
ORDER BY programme.date, programme.start
```

Is Rutger Hauer on any channel tomorrow after five pm?

22 STATEMENT TEMPLATE

user has given CHANNEL, TIME, empty ACTOR/DIRECTOR/PRESENTER

user expects TITLE, CHANNEL, START and DATE

```
SELECT title.title, credits.<creditsAspect>,
programme.channel, programme.start, programme.date
FROM title, programme, credits
WHERE title.ForeignKey = programme.PrimaryKey
AND programme.channel = <channel>
AND programme.start = <start>
AND programme.date = <date>
AND credits.ForeignKey = programme.PrimaryKey
ORDER BY programme.date, programme.start
```

Who is acting on Channel5 tonight?

23 STATEMENT TEMPLATE

user has given empty CHANNEL, TITLE

user expects TITLE, CHANNEL, START and DATE

```
SELECT title.title, programme.channel,
programme.start, programme.date
FROM title, programme
WHERE title LIKE <title>
AND programme.PrimaryKey = title.ForeignKey
ORDER BY programme.date, programme.start
```

What channel shows the Simpsons?

24 STATEMENT TEMPLATE

user has given CHANNEL, CATEGORY, ACTOR/DIRECTOR/PRESENTER

user expects TITLE, START and DATE

```
SELECT title.title, programme.start, programme.date
FROM title, programme, category, credits
WHERE title.ForeignKey = credits.ForeignKey
AND programme.channel = <channel>
AND category = <category>
AND credits. creditsAspect LIKE <creditsValue>
AND category.ForeignKey = title.ForeignKey
AND credits.ForeignKey = programme.PrimaryKey
ORDER BY programme.date, programme.start
```

Are there any movies with Shannon Tweed on BBC1

25 STATEMENT TEMPLATE

user has given CHANNEL, TITLE

user expects START and DATE

```
SELECT programme.start, programme.date
FROM title, programme
WHERE title LIKE <title>
AND channel = <channel>
AND programme.PrimaryKey = title.ForeignKey
ORDER BY programme.date, programme.start
```

Is Simpsons on BBC2?

Appendix C: Sample Dialogue Collection

The following utterances are examples of user input to the dialogue system prototype.

1. What shows are there tonight?
2. What programs are on tomorrow night?
3. What's on right now?
4. Show me all programs currently screening.
5. Could you please tell me what's on right now?
6. Please tell me what sports events are on tonight.
7. Are there any movies tomorrow after 7 pm?
8. Show me when the next news show with weather is.
9. What movies are showing this week?
10. Show all movies starting after 7 pm for Monday, Wednesday and Friday next week.
11. Which channel shows sitcoms at 5 pm tonight?

12. Where can I watch movies tomorrow?

13. What channels feature cultural programmes?

14. Is there a news channel?

15. Which channel is "Friends" on tonight?

16. Is there a channel showing "Braveheart" this week?

17. I wanna watch "60 Minutes". Is it on tonight?

18. I wanna watch "60 minutes". What channel is it on and when?

19. Which channel shows "60 minutes"?

20. When is the next episode of Star Trek?

21. Who is the main actor in "Living Daylights"?

22. Who directed "Gladiator"?

23. When was "Star Wars" produced?

24. Who is the director of "Jailhouse Rock"?

25. Tell me more about tonight's Ricki Lake show.

26. Can you tell me what "Return of the Jedi" is about?

27. Show me the synopsis for The Quest for the Holy Grail.

28. Who is presenting "America's Dumbest Home Viedeos"?

29. What's on NaturePlanet tomorrow at 8 pm?

30. Show me all CNN shows tonight.

31. I'd like to know what movies are featured on BBC2 on Friday.

32. Is there a movie with Nick Nolte tonight?
33. I wanna watch a movie with Nicole Kidman this week.
34. Are there any movies directed by George Lucas tomorrow?
35. Are there any movies starring Mel Gibson on Channel5 next week?
36. Who's the main actor in tonight's 8 o'clock movie on BBC2?
37. Is Gladiator a movie?
38. Is Superbowl a movie?
39. What is on BBC1 before 5 pm?
40. Is Superbowl on BBC2 tomorrow?
41. When is The Simpsons?
42. What channels do I have?
43. Where can I watch series tomorrow?
44. Is there a news channel?
45. Who has directed Gladiator?
46. Who is the director of Gladiator?
47. Is there any information about who the director of Gladiator is?
48. Who directed Gladiator?
49. What is Gladiator?