Linköping Studies in Science and Technology

Thesis No. 890

Domain Knowledge Management in Information-providing Dialogue Systems

by

Annika Flycht-Eriksson

INSTITUTE OF TECHNOLOGY

LINKÖPINGS UNIVERSITET

Submitted to the School of Engineering at Linköping University in partial fulfilment of the requirements for the degree of Licentiate of Philosophy

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2001

Domain Knowledge Management in Information-providing Dialogue Systems

by

Annika Flycht-Eriksson

May 2001
ISBN 91-7373-050-5
Linköpings Studies in Science and Technology
Thesis No. 890
ISSN 0280-7971
LiU-Tek-Lic-2001:27

ABSTRACT

In this thesis a new concept called **domain knowledge management** for information-providing dialogue systems is introduced. Domain knowledge management includes issues related to representation and use of domain knowledge as well as access of background information sources, issues that previously have been incorporated in dialogue management.

The work on domain knowledge management reported in this thesis can be divided in two parts. On a general theoretical level, knowledge sources and models used for dialogue management, including domain knowledge management, are studied and related to the capabilities they support. On a more practical level, domain knowledge management is examined in the contexts of a dialogue system framework and a specific instance of this framework, the ÖTRAF system. In this system domain knowledge management is implemented in a separate module, a **Domain Knowledge Manager**.

The use of a specialised Domain Knowledge Manager has a number of advantages. The first is that dialogue management becomes more focused as it only has to consider dialogue phenomena, while domain-specific reasoning is handled by the Domain Knowledge Manager. Secondly, porting of a system to new domains is facilitated since domain-related issues are separated out in specialised domain knowledge sources. The third advantage with a separate module for domain knowledge management is that domain knowledge sources can be easily modified, exchanged, and reused.

This work has been supported by The Swedish Transport & Communications Research Board (KFB).

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Acknowledgements

First and foremost I would like thank my supervisor Arne Jönnsson who with contagious enthusiasm has guided me in the work that has resulted in this thesis. Despite a very busy schedule he has always been available for questions and discussions, and he has read and commented on a great number of drafts of this thesis. I am very grateful for his supervision, without it I would not have made it. I would also like to thank my secondary supervisors, Nils Dahlbäck and Lars Degerstedt, who through insightful discussions and comments have helped me bring order to my thoughts and the material presented in this thesis. Thanks also to the other members of NLPLAB, who have contributed to the very stimulating and positive atmosphere this thesis has been created in.

When I have had problems with unco-operative computers or questions regarding technical matters Bernt Nilsson has helped me, many thanks for this. Thanks also to Marie Ekström Lorentzon, Lise-Lott Andersson and Lillemor Wallgren who have handled the administration. Finally, I would like to thank Ivan Rankin for improving my English.

Contents

Chapter 1

Introduction

The topic of this thesis is domain knowledge management in information-providing dialogue systems. The purpose of the dialogue in such systems is to help the user formulate information requests that the system can respond to by retrieving information from one or several information sources and presenting it to the user. The information provided by the system is usually restricted to one specific domain, for example travel or weather information.

To make the interaction natural the system needs to represent and reason about features of the domain. Consider, for instance, an information-providing dialogue system in the domain of local public transportation. To respond intelligently to a user utterance such as "I want information about buses to Vidingsjö in the evening", the system needs to represent and reason about geographical and temporal features of the domain. Geographical knowledge is needed to detect that "Vidingsjö" is the name of a suburb and to find some suitable bus stops in this suburb. Temporal knowledge is required to handle the expression "in the evening", which in the context of local public transportation should be interpreted as the time interval from 6 p.m. to 10 p.m. This thesis explores how such domain knowledge

can be represented and used to support capabilities that are needed to achieve natural interaction with a dialogue system.

1.1 Dialogue systems

The term **dialogue system** has been used to denote very different types of systems. The disparity is due to the twofold heritage of the dialogue system research area. On one hand, there is the *speech* community where researchers have been working on combining the achievements of speech recognition and speech synthesis research, thus creating spoken language systems. On the other hand, the *computational linguistics* and *artificial intelligence* communities have been working on the modelling of language, discourse and dialogue. Work that has often resulted in implementations of natural language systems with typed input and output.

This twofold legacy poses problems when trying to define what a natural language dialogue system is, because depending on their background, researchers will give very disparate answers both on what characterises a dialogue system and which aspects to model in the architectures. I use the term **dialogue system** to denote a computer system that interacts with users utilising connected natural language dialogue, where the use of language need not consist of predefined commands.

Interactive dialogue systems have been developed since the late 1980s. The PHILIPS dialogue system [34] for train timetable information in Germany was the first commercial system to be made publicly available in 1994. The PHILIPS system has since then been customised for other countries and languages, for example The Netherlands and Sweden. Figure 1.1 shows a dialogue with the German system (translated from German to English).

Today there are several information-providing dialogue systems on the market and they are becoming increasingly common. Other types

S1: Good morning. This is the automatic timetable information service. How can I help you?
U2: Good morning. I need a connection from Hamburg to Munich.
S3: When would you like to go from Hamburg to Munich?
U4: Next Friday, er, at eight o'clock.
S5: So you want to travel on Friday the 23rd of September at 8 a.m.?
U6: Exactly.
S7: There is the following connection: with Intercity Express 793, departure from Hamburg at 8:02, arrival in Munich at 14:05. Would you like me to repeat the connection?
U8: No but I need a train for the return trip on Sunday.
S9: At what time would you like to go from Munich to Hamburg on Sunday the 25th of September?
U10: I have to arrive in Hamburg late in the evening.
S11: There are three connections. This is the first one: with Intercity Express 880, departure from Munich at 15:54, arrival in Hamburg at 21:58. Would you like me to repeat the connection?
U12: No, thank you.
S13: Would you like the next connection?
U14: No, thanks, this one is fine.
S15: Do you need another connection?
U16: No.
S17: I wish you a pleasant journey. Thank you for calling. Good bye.

Figure 1.1: A dialogue between a user and the system for train timetable information developed by Philips [9].

of dialogue systems, like problem-solving, argumentation, advisory and tutoring systems, have also been developed but have not been commercialised to the same extent. An example of a problem-solving research system is the well-known TRAINS [25] system in which the system and a user co-operate to plan railroad transportations. An example dialogue is given in Figure 1.2.

U1: I need to get to Detroit, Syracuse and Montreal.
S2: Ok. [highlights the three cities]
U3: Let's send the trains from Chicago to Detroit.
S4: Ok. [draws route from Chicago to Detroit via Toledo] But the terminal at Toledo is delaying traffic due to bad weather. trains will take an additional five hours to move through there. [highlights Toledo red]
U5: That's ok. Now let's go from Atlanta up to Syracuse.
S6: Please tell me a route to use to get from Atlanta to Syracuse. [highlight engine at Atlanta and city Syracuse]
U7: Go via Charleston and Pittsburgh.
S8: Ok. [draws route via Cincinnati] But traffic through Cincinnati is being delayed due to construction.
U9: Ok, lets go via Charleston and Buffalo instead.
S10: No problem. [draws route]
U11: That's great.
S12: I hope it was helpful.
U13: Finally, I want to go to Montreal from Philadelphia.
S14: Ok. [draws route]
U15: That's great. I'm done.

Figure 1.2: A dialogue between a user and the TRAINS system.

As title of this thesis implies, it deals primarily with dialogue systems that provide information, although problem-solving systems are also discussed. **Information-providing** systems are systems where the user wants some information that is available in one or more background systems. The system helps the user to construct an information request that is specific enough for the system to retrieve the

information and present it to the user. Such information-providing dialogue systems are also called *simple service systems* [35]. The PHILIPS train information system [34] is one example of such a system. **Problem-solving** systems collaborates with the user to solve tasks that he or she cannot solve alone. Usually, the human knows about strategies and the system has information about the problem and various details. Thus, it is very important that the system and user co-operate to make the best of their specialities. TRAINS [25] is a typical problem-solving dialogue system.

1.2 Knowledge sources in dialogue systems

Research on dialogue systems can be divided in three areas: interpretation, dialogue management, and generation. Each of these areas utilise knowledge sources of various types, such as lexicons, grammars, dialogue models etc. Since the focus is on domain knowledge management, knowledge sources used for interpretation and generation is not considered in this thesis. Dialogue management has, however, often incorporated aspects of domain knowledge management and therefore knowledge sources used in this area will be discussed.

The term **knowledge source** will be used to denote a component in a dialogue system, which consists of a knowledge model and mechanisms used to manipulate and reason about the information held by the model. The types of knowledge sources used for dialogue management in dialogue systems are related to knowledge of dialogue and discourse, task, user and domain.

Knowledge of various features of **dialogue and discourse**, such as turn-taking, grounding, topic and referring expressions, is of course crucial to a dialogue system. Knowledge sources for dialogue knowledge can be used for various purposes; the most prominent is to decide what is an appropriate response to a user utterance, for example, if a clarification should be initiated, e.g. S6 in figure 1.2, or database

access and presentation of information as in S7 in figure 1.1. They can also be used to make context-dependent interpretations, as in U8-S9 in figure 1.1 where a return trip is interpreted as a trip from Munich to Hamburg.

The **tasks** performed by dialogue systems differ in character depending on the service-type of the system. In information-providing systems the tasks are communicative, mainly to exchange information, for example the system asks the user to specify parameters of a request or presents requested information. Problem-solving systems deal with tasks that take place outside the system and are executed or planned during the dialogue. Knowledge sources with task knowledge can help the system in the interaction with the user. The PHILIPS system can, for example, use knowledge of the system's task of providing train information to decide which information to ask for from the user, e.g. S3 in figure 1.1.

Knowledge of the **user** can also improve the quality of the interaction. The TRAINS system, for example, utilises a user model to decide what information to present. In S4 and S8 in figure 1.2 the system provides information that is relevant for the task and previously unknown by the user.

To be able to conduct a dialogue and perform a task the system has to have knowledge of the world that is under discussion. However, to make a feasible system the knowledge of the world has to be restricted to some aspects that are useful for the task at hand; this restricted view of the world is the **domain** of the system. A domain can be defined as "a section of the world about which we wish to express some knowledge" [55].

Knowledge of the **domain** can be represented in various ways. I make a distinction between conceptual models and domain knowledge sources. A conceptual model is often a representation of a set object types or concepts, their properties and relations among them. For example, a conceptual model can represent information on how the concept 'velocity' is related to the concept 'car'. Domain knowledge sources contain a data- or knowledge-base and reasoning mechanisms

for manipulation of this, for example, a temporal knowledge source can include a calendar and temporal reasoning mechanisms. A difference between domain knowledge sources and conceptual models is that domain knowledge sources often contain a subset of general world knowledge, while the conceptual model can represent application-specific deviations from the general meaning of a concept.

In relation to domain knowledge sources, background and application systems should be mentioned. A background or application system is the primary information source in an information-providing dialogue system, or the problem-solving component in a problem-solving dialogue system. Domain knowledge sources are often needed to support access of the background system, for example by mapping vague expressions to a more suitable format. For instance, to correctly interpret the utterances U4, U8, and U10 in the dialogue with the PHILIPS system (figure 1.1), it has to have knowledge of and be able to reason about the temporal expressions that describe dates and times. In U4 the expressions "next Friday" and "at eight o'clock" have to be mapped to precise entities, in this case the 23rd September and 8 a.m. A similar problem is present in U8 where "Sunday" should be mapped to a date. In this context of a return trip "Sunday" means the Sunday following the Friday on which the trip begins. The system, thus, needs both domain knowledge and knowledge of the context, i.e. the previous exchanges between the user and the system. Finally, in U10 the vague description "late in the evening" must be mapped to a time interval, thus requiring temporal knowledge.

Examples of the need for domain knowledge and reasoning can also be found in the TRAINS dialogue (figure 1.2). The system must have knowledge about possible routes between cities, and situations that are potential problems along the routes. For example, in S4 and S8 the system evaluates the routes proposed by the user and points out possible delays due to bad weather and construction.

Although knowledge of dialogue, task, user, and domain has been presented as separate entities, there is often a mixture of knowledge types in the knowledge sources used in dialogue systems, for example, task, domain and dialogue knowledge are often integrated.

This mixture of knowledge has a number of drawbacks, especially for research systems and systems not primarily developed for limited dialogue in one application. First of all it is hard and time-consuming to port a system to a new domain or task. Another related problem is that it is difficult to experiment with a system's behaviour, for example trying different dialogue strategies, since a change in some item often causes other changes. The lack of clear boundaries between models also makes it hard to reuse and incorporate previous work done by others. These problems indicate that clear definitions of the different models are desired.

One objective with the work presented here is therefore to clarify the situation by characterising the knowledge sources, their roles and relations.

1.3 Domain knowledge management

Development of a usable dialogue system requires considerable effort. An important aspect when developing a dialogue system is therefore portability; to be able to easily customise the dialogue system for a new task or domain. Recently the idea of portability has been taken further and development of frameworks and toolkits that can be used as the basis for development of new dialogue system has been promoted. CSLUrp [18] is an example of a toolkit that can be used to construct simple dialogue systems, TRINDIKIT [66] is a more sophisticated toolkit for information-providing dialogue systems, and TRIPS [6] is a generalisation of TRAINS to a framework for problem-solving dialogues.

A prerequisite for development of dialogue system frameworks is that domain-dependent features can be separated from domain-independent features. Furthermore, it must be easy to incorporate various background systems and domain knowledge sources in a dialogue system that has been based on the framework. A way to achieve this is to separate management of domain knowledge and the background systems from dialogue management, which leaves the dialogue management more focused and domain-independent.

A second objective of this thesis is to examine how this separation can be done. A suggestion of how domain knowledge management can be clearly separated from dialogue management is presented. A new module, a Domain Knowledge Manager, which is responsible for domain knowledge sources and background systems is introduced and it is shown how this new module manages the domain knowledge.

1.4 Issues and contributions

The work presented in this thesis can be divided in two main parts. The first part concerns the relations between various types of knowledge sources used for dialogue management in dialogue systems, and the capabilities they support.

Issues:

- What characterises the knowledge sources commonly used in dialogue systems? What roles do the different knowledge sources and models have? What are the relations between the different knowledge sources and models?
- What capabilities can help a dialogue system achieve natural interaction with a user? What knowledge sources and models are required for the different capabilities?

Contributions:

- A characterisation and categorisation of the various types of knowledge sources and models used in dialogue systems, which contributes to a clarification of the sometimes confusing terminology. This is based on a survey of several information-providing and problem-solving dialogue systems.
- A mapping of the knowledge sources and models to dialogue system capabilities considered useful to achieve natural interaction. The dialogue system capabilities are compiled from a set of guidelines and development principles for dialogue systems.

The second part of the work presented in this thesis focus on domain knowledge management in a framework for information-providing dialogue systems.

Issues:

- What are the desirable capabilities of an information-providing dialogue system?
- How is domain knowledge management related to dialogue management? How can domain knowledge management be separated from dialogue management?
- How can domain knowledge management be realised in a dialogue system?

Contributions:

- An empirically based set of capabilities required for information-providing dialogue systems. This set of desirable capabilities is the result of a corpus study.

- A characterisation of dialogue management and domain knowledge management in terms of the capabilities they should provide and the knowledge sources used to support the capabilities. The relations between the capabilities and the knowledge sources were used as a basis for distinguishing dialogue and domain knowledge management capabilities.
- A separate module for domain knowledge management, a Domain Knowledge Manager, to be used in information-providing dialogue systems. The design of the module is made in the context of a dialogue system framework called MALIN. The Domain Knowledge Manager was is also implemented in a dialogue system that provides information on local public transportation, the ÖTRAF system.

1.5 Thesis outline

The topic of this thesis is domain knowledge management in information-providing dialogue systems. Aspects of domain knowledge and background and application systems have, however, often been incorporated in dialogue management. The first two chapters are therefore concerned with knowledge sources and models, and dialogue system capabilities related to dialogue management. In the following chapters the focus then shifts towards domain knowledge management which is presented and discussed in the context of a dialogue system framework and a particular instance of this framework, an information-providing dialogue system in the domain of local public transportation. A more detailed description of the chapters follows below.

Chapter 2 **Knowledge sources for dialogue management** The knowledge sources and models used in information-providing and problem-solving dialogue systems are characterised and their roles and relations in dialogue systems are discussed.

Chapter 3 **Capabilities for dialogue management** To achieve natural and graceful interaction with a user, a dialogue system must have a variety of capabilities. A list of such capabilities is presented with a discussion on how the capabilities are related by the knowledge sources and models presented in chapter 2.

Chapter 4 **Capabilities for information-providing dialogues** The result of an empirical study of corpus material used to decide which capabilities an information-providing dialogue system requires is presented. The capabilities are discussed in terms of the knowledge they require.

Chapter 5 **Domain knowledge management in the MALIN framework** A framework for information-providing dialogue systems, MALIN, which includes a new separate module for domain knowledge management, a Domain Knowledge Manager is presented. The architecture and knowledge sources of the Domain Knowledge Manager are presented and discussed.

Chapter 6 **Domain knowledge management in the ÖTRAF system** A specific instance of the MALIN framework, the ÖTRAF System, for the domain of local public transportation is presented. Different features of domain knowledge management are exemplified and clarified.

Chapter 7 **Summary and discussion** The concept of domain knowledge management and the proposed design of a Domain Knowledge Manager are summarised and discussed.

Chapter 2

Knowledge sources for dialogue management

There is a variety of dialogue systems that provide information services or assistance in solving a specific task. The systems differ in complexity due to the domain and the approach taken in the design of the system. Some systems are highly knowledge-intensive and contain several interacting knowledge sources and models, while others rely on much simpler models and procedures. The variety of dialogue system architectures that incorporate various models has led to confusion when it comes to the purpose and contributions of a specific model. The relations between various knowledge sources and models are also diffuse. In order to clarify the situation, the following issues are examined in this chapter[1]:

- What characterises the knowledge sources and models used for dialogue management in dialogue systems?
- What roles do the different knowledge sources and models have?

[1]This chapter is a revised and extended version of [26]

- What are the relations between the different knowledge sources and models?

To address these issues a study has been conducted on how knowledge sources and models are utilised in eight different information-providing or problem-solving dialogue systems: CIRCUIT FIX-IT SHOP, GALAXYII, LINLIN, RAILTEL, SUNDIAL, TRAINS, VERBMOBIL, and WAXHOLM[2]. The survey of knowledge sources and models focuses on knowledge for dialogue, discourse, task, domain and users. Although there are other models, this selection represents the most common types of knowledge sources and models utilised in information-providing and problem-solving dialogue systems today.

2.1 An overview of the surveyed dialogue systems

Dialogue systems are developed for very different tasks and domains. In this survey the focus is on two types of dialogue systems, *information-providing* systems and *problem-solving* systems[3], leaving out other types of systems such as argumentation and tutoring systems. In this survey the GALAXYII, LINLIN, RAILTEL, SUNDIAL, and WAXHOLM systems belong to the information-providing category. The CIRCUIT FIX-IT SHOP and TRAINS systems are instances of the problem-solving class of systems.

[2]The systems have been chosen to cover most types of dialogue systems: commercial or research, plan-based or grammar-based, general purpose or domain-specific, and natural language only or multi-modal

[3]Since there is no consensus on classifications of dialogue systems, I have made a distinction between information-providing and problem-solving systems, both of which are sometimes refered to as task-oriented.

2.1.1 Circuit Fix-it Shop

The Circuit Fix-it Shop system is an example of a problem-solving system that assists a user as he or she is fixing an electric circuit. The system is based on the Missing Axiom Theory [61] in which completion of an action is viewed as a theorem and the process of accomplishing an action or a task is the same as proving the theorem. If a task cannot be completed, it is interpreted as a missing axiom which has to be provided by the user through a dialogue with the system.

The system consists of five components: a dialogue controller that is the supervisor of the system, a domain processor that holds information about the application domain, a general reasoning component responsible for theorem proving, a linguistic interface component for interpretation and generation of utterances, and finally a knowledge module which represents knowledge about the dialogue, the user and the actions that can be performed.

2.1.2 GalaxyII

The Galaxy system, followed by the GalaxyII system, developed by the Spoken Language Systems group at MIT, is a distributed multi-modal multi-domain system that provides users with information about, for example, travel and weather [30]. The GalaxyII system consists of several specialised modules, e.g. for speech recognition and understanding, dialogue management and context tracking, application back-ends, generation and speech synthesis. The different modules interact via a hub using a common knowledge representation format, semantic frames [59].

2.1.3 LINLIN

The LINLIN system[4] [2] is a natural language dialogue system that has been customised for various domains, e.g. information on second-hand cars and charter trips to the Greek archipelago. The system includes a generator, a parser, a database interface, an instantiator, and a dialogue manager that is the kernel of the system. The dialogue manager is responsible for controlling the interaction with the user and also controls the other modules.

2.1.4 RAILTEL

The RAILTEL system is an example of a practical system that is publicly available and currently in use [8]. It provides information over the telephone about railway journeys. The system is composed of a number of components, such as a speech recogniser and a natural language component, a dialogue manager, and a response generator. The architecture is serial: information represented as semantic frames flows from the speech recogniser to the language understanding module, on to the dialogue manager which accesses the application and forwards the result to the response generator.

2.1.5 SUNDIAL

The SUNDIAL system has been developed within the SUNDIAL project (Speech UNderstanding in DIALogue) where several dialogue systems for information exchange over the telephone have been created for different languages [36]. The system provides information over the telephone about air and train traffic. The system contains a component for speech recognition, a parser, a dialogue manager, a text

[4] LINLIN is strictly speaking a framework that can be used to implement various dialogue systems. The name has, however, also been used to denote some customised systems, and this is how it will be used in this chapter.

generator, and a text-to-speech system. The dialogue manager in the SUNDIAL system has a distributed architecture, and consists of five modules: a Linguistic Interface, a Message Generator, a Dialogue Module, a Belief Module, and a Task Module, which can be seen as independent agents. Its purpose is to interpret the input that has been analysed by the parser, decide how to continue the dialogue, and plan the system utterances [10].

2.1.6 TRAINS

The TRAINS system is used for mixed-initiative collaborative planning in the transport domain [7]. It is task-oriented, giving the user advice on how to perform a task in the real world outside the system. The system is designed as an agent having a mental state and the dialogue management is plan-based. It is based on the common BDI (Beliefs Desires Intentions) model which have been adapted to conversation between two agents. The first system TRAINS-90 has over the years been redesigned and improved and there are a TRAINS-93 [64], TRAINS-95 [24] as well as a TRAINS-96 [25] system. The TRAINS-96 system consists of a number of modules for interpreting and generating natural language, dialogue processing and domain reasoning. The modules are connected in a star-like fashion and message passing is handled by a specific processor. KQML (cf. [67]) is used as the communication language.

2.1.7 WAXHOLM

The WAXHOLM system provides users with information about boat traffic in the Stockholm archipelago. It has much in common with RAILTEL, both systems are domain-specific spoken dialogue systems originating from the speech research community. However, the WAXHOLM system also allows for limited multi-modal interaction [11]. The interaction between modules within the system is based on the use of semantic frames. The module for speech recognition and interpre-

tation delivers a semantic frame representation of a request to the dialogue manager, which decides how to respond to the request and updates the semantic frame. The updated frame is then sent to the components responsible for generation of speech and graphics.

2.1.8 VERBMOBIL

The VERBMOBIL system is not a traditional dialogue system as the system is not an active participant in the dialogue. Instead, it listens in on two persons having a conversation in a language that is not their mother tongue, e.g. a German and a Japanese trying to book a meeting speaking English. The system's task is to monitor the dialogue and to give the users translations when required [4].

The system has two different processing modes. When the dialogue participants are speaking English and no translation is necessary, only shallow processing takes place. This means that the system only looks for keywords and tries to identify the speech act performed. In this way, the system knows at what stage the dialogue is. When a translation is requested, the system enters a mode of deep processing where utterances are analysed with respect to prosody, syntax, semantics and pragmatics. Besides modules for keyword spotting, speech recognition and language analysis, there are modules for semantic construction and evaluation, transfer, and generation [3].

2.2 Characteristics of knowledge sources

Even if dialogue system architectures differ all systems need and utilise similar knowledge. However, the terminology used to describe the knowledge sources and models varies between the systems. Therefore a uniform terminology is introduced in this section, and the various knowledge sources and models used by the systems are characterised in terms of the type of knowledge they represent.

2.2.1 Discourse and dialogue knowledge

In dialogue systems two aspects of the dialogue need to be modelled, a generic description of the structure of dialogues that can be used to form a coherent dialogue with the user, and a dynamic representation of the current dialogue. I will refer to the first as the dialogue model and the second as the dialogue history. The dialogue model is often closely connected to and dependent on the information represented in the dialogue history.

Dialogue models have the common purpose of describing how the system should respond to user utterances, for example by accessing a database or asking for clarifications. The general information about the dialogue, held by the dialogue model, is often based on a representation of relations between the constituents of a dialogue. This knowledge is used to control the interaction, i.e. to decide what action to take in a certain situation.

There are several approaches to dialogue modelling used today: the most common are state transition networks, grammar-based, and plan-based approaches.

State transition networks are a rather simple method for modelling and controlling dialogue flow. The system responds to a user utterance by moving from one state to another, thus traversing the network and producing a sequence of exchanges between the user and the system. The advantage of this type of model is its simplicity, given that the task being modelled is well-structured and consists of a limited number of necessary exchanges. If the task is highly complex, the transition networks tend to get unmanageable. This is also true if the model should contain repair strategies and allow the user more unrestricted input [48].

Dialogue grammars have a rather long history and are based on the notion of adjacency pairs [56]. Adjacency pairs express the fact that speech acts typically form a regular sequence, such as a question followed by an answer. Rules in the dialogue grammar capture the

sequential and hierarchical constraints of dialogues in the same way that grammar rules describe the syntactic structure of a sentence.

Plan-based models go beyond the utterance and try to model the speaker's intentions and goals [13]. Communication becomes a part of the speaker's overall behaviour. Elements in these models are plans, actions, mental states and mechanisms for recognising a specific plan and for reasoning about the speaker's beliefs, intentions, and actions.

The dialogue models used by the systems in the survey incorporate both grammar-based and plan-based dialogue modelling. The RAILTEL, SUNDIAL, LINLIN, and WAXHOLM systems have a dialogue model consisting of a *dialogue grammar* that models the dialogue's hierarchical structure. In the RAILTEL system the dialogue is partitioned into three phases each consisting of a number of sub-dialogues. It is modelled by a hierarchical structure of sub-dialogues and dialogue acts, represented by a set of rules in a dialogue grammar [8]. A similar approach is taken by SUNDIAL [10] and the LINLIN system [39], which model the dialogue using a dialogue grammar and speech acts.

The dialogue model in the WAXHOLM system is based on the idea that the dialogue should be described as a grammar and at the same time be probabilistic [12]. The WAXHOLM system thus has a dialogue grammar represented as finite state machines but also a *statistical* component for topic prediction, which is part of the dialogue model.

A combination of approaches is also utilised in the VERBMOBIL system. The possible moves for the user and system to make during the interaction are represented in a dialogue model using dialogue acts. The dialogue module in the VERBMOBIL system consists of three submodules: a Statistical Module, a Finite State Machine, and a Dialogue Planner. The Statistical Module can, given a dialogue state, predict all possible successive dialogue acts and their likelihood. The Finite State Machine has a similar task: it checks whether a dialogue act is consistent with the underlying dialogue model and which subsequent dialogue acts are possible. The Dialogue Planner is the most sophisticated of the three sub-modules. Plan operators are utilised to plan the dialogue and handle different phases like negotiations,

clarifications and repairs. A plan is built up hierarchically with the leaves in the hierarchy being dialogue acts [4].

The TRAINS system takes a distinct *plan-based* approach to dialogue modelling. The system is viewed as a conversational agent having goals, intentions, plans, and obligations. Answers to user utterances are planned by the system in order to reach its goals and at the same time fulfil its obligations [64].

Another way to handle the dialogue is to do it more procedurally as in the GALAXYII system where the dialogue is controlled by a stepwise process. This means that there is no explicit dialogue model as the model lies implicit in the processing algorithm [58].

The CIRCUIT FIX-IT SHOP system also lacks an explicit dialogue model; instead handling the dialogue is tightly coupled to the task model. The task model is used to find missing axioms that result in clarification dialogues and also to insert sub-tasks corresponding to sub-dialogues initiated by the user [61].

The dialogue model is often used together with information about the dialogue state. The variety of ways to model the state of the dialogue has led to a variety of names, e.g. discourse memory, discourse history, dialogue memory, dialogue history, history table, and context model. To avoid confusion only the term dialogue history will be used. Dialogue histories could then be described as partial or full depending on the number of information levels, and the degree of structure could also be used for further classification.

The dialogue history in a dialogue system represents the state of the dialogue, that is, what has been talked about and the current topic of the dialogue. It can be used for dialogue control, disambiguation of context-dependent utterances, and context-sensitive interpretation, e.g. reference resolution and handling of ellipsis.

The representations of dialogue histories range from complex hierarchical structures, containing a variety of information, to much simpler sequential representations. The kind of dialogue history required in

a dialogue system depends on the linguistic phenomena that have to be handled, such as misunderstandings, interruptions, and deictic expressions, and the complexity of the task and domain which is reflected in the interaction. If a dialogue is made up of several clearly separated sub-dialogues, a more structured representation may be preferred to a sequential.

In the LINLIN system a dialogue tree consisting of dialogue objects is constructed during the interaction [39]. The tree has three levels which correspond to the whole dialogue, discourse segments called initiative response units, and dialogue moves. A dialogue object contains situation parameters and content parameters. The first holds information about Initiator, Responder and context parameters while the second records the current focus of the dialogue.

The representation of the dialogue history in the SUNDIAL system is distributed over several models. A tree is used to represent the dialogue structure. It resembles the dialogue tree in the LINLIN system but has one more intermediate level to represent a common topic [10]. An interpretation is called a belief and a sequence of beliefs which corresponds to the user's utterances is represented in a contextual model [47]. This model is used to make context-sensitive semantic interpretations of user utterances. Changes in the contextual model are reflected in a flag called status that indicates how the contextual model has changed; the value repeat, for example, means that nothing new has been contributed. This information can be used by the dialogue module to reason about the function of an utterance, e.g. if it is a confirmation or a new request [36].

The dialogue planner in the VERBMOBIL system contains a dialogue history representing intentional, thematic and referential information. Intentional information corresponds to the dialogue acts performed by the participants, and is structured in a tree-like representation. The thematic information refers to relevant information in utterances while referential information is the lexical realisation of utterances. A representation of the proceeding dialogue is constructed dynamically in the dialogue memory during the dialogue [4].

Some of the other systems focus only on the objects that have been mentioned, which could be compared with the attentional level of the discourse using Grosz & Sidner's terminology [32]. The GALAXYII, RAILTEL, and WAXHOLM systems all maintain a history of semantic frames that are used to represent the objects and relations in focus. The discourse module in the GALAXYII system maintains a history table of referable objects to facilitate the interpretation process. Items and requests are represented internally as semantic frames, and the history table consists of a sequence of frames [58]. In the RAILTEL system the context is represented by a dialogue history and a generation history which contain the semantic frame representation of the user's and the system's previous utterances respectively [8]. The utterances are stored sequentially within these. The WAXHOLM system utilises a dialogue history based on the semantic frame representation and the updates of this representation [12].

The dialogue history in the TRAINS system lies somewhere between the complex multi-level and simple frame-based representations, as it models both attentional and intentional features. The discourse is modelled as a stack of discourse units (DUs) which represent utterances and the corresponding speech acts. In the model the initiator and state of DUs are also recorded. Two other contextual features represented in the system are the turn and the local initiative, each of which is held by one of the dialogue participants. The turn indicates who is speaking now, while the local initiative informs which speaker is obliged to speak next [64].

The CIRCUIT FIX-IT SHOP system differs from the other systems as it focuses on what might be said next instead of what has been said earlier. Thus, the attentional state is represented as a set of expectations about the possible responses from the user [60].

2.2.2 Domain knowledge

The amount of domain knowledge needed in a dialogue system differs depending on the domain and the system's task. This means

that the sources of domain knowledge can range from rather simple conceptual models to full-fledged world models capable of complex reasoning. Dahlbäck & Jönsson [22] make a distinction between domain models and conceptual models. The domain model represents the structure of the world and usually comprises a subset of general world knowledge, while the conceptual model represents the general and domain-specific concepts. I will also use this distinction, but since reasoning is central to what Dahlbäck and Jönsson call domain models I will instead use the term domain knowledge source.

Information from the conceptual model and the domain knowledge sources is primarily used to identify the relevant items and relations that are discussed, reason about and derive new information from the information provided by the user, to supply default values, etc. In information-providing systems the knowledge represented in a domain knowledge source is often closely connected to the background system, e.g. a database system. In such cases domain knowledge is used to map information in the user's utterances to concepts suitable for database search.

The semantic frames used in the RAILTEL system contain the relevant domain concepts and serve as a simple conceptual model. The domain knowledge also consists of two kinds of rules: *Default value rules* supply default values, e.g. the current or next month for a departure date; *Interpretative rules* transform vague qualitative values into more precise quantitative values used by the system, for example they map the concept "morning" onto the precise interval "between 6 a.m. and noon" [8].

The domain knowledge in the SUNDIAL system is distributed. One module contains a hierarchy of surface-oriented concepts while another module contains a hierarchy of task-oriented concepts. The surface-oriented concepts, for example "at noon", are mapped onto task-related concepts, "at twelve o'clock". The domain knowledge thus consists of two concept hierarchies and inference rules used to map one to the other [36].

Domain knowledge in the VERBMOBIL system is represented in a conceptual hierarchy that represents relations between different categories, entities, and situations. Situations are entities determined by spatial and temporal features like year, month, week, day, location, etc. The hierarchical structure of the model makes it suitable for decisions about possible references. It also allows inheritance; if a temporal object is available for scheduling, the super-ordinate object is regarded as free too, and similarly, if an object is not available, none of the subordinated objects are either [51].

The TRAINS system on the other hand has a more complex representation of domain knowledge. The Domain Plan Reasoner in the system is responsible for the representation of and reasoning about the domain. It maintains a representation of the state of the world, provides new suggestions about routes, and adjusts the current plan given new constraints. Inspection of plans that have been suggested by the user is also done to check whether any unknown conditions need to be considered [24].

The CIRCUIT FIX-IT SHOP system also needs substantial knowledge about the domain. The domain knowledge is divided into a conceptual model and a domain model. The conceptual model represents how different components of a circuit are related to each other and how the different components should function. The domain model consists of a set of axioms that represent the current state of the circuit that is being fixed [60]. The term domain model thus has a different meaning in this system since it is more of a representation of a state than general knowledge about the domain.

In the GALAXYII system conceptual models and domain knowledge sources are combined for some domains. The domain knowledge in the GALAXYII system can be found in two places, declarative tables in the discourse module, and in domain servers that contain the application data. The tables can be seen as conceptual models which describe the possible semantic classes a value can have and some relations between them. The domain servers can contain more specific domain knowledge needed to interpret the semantic frame [58].

In the LINLIN system a conceptual model has been used in the travel domain. It consists of a hierarchy with concepts for charter trips, such as resort, hotels, etc. It is used for focus tracking and to resolve anaphoric expressions [22].

The WAXHOLM system contains no explicit domain knowledge sources; domain knowledge is instead incorporated in the lexicon and the grammar. The parser is based on a semantic feature structure with features of two kinds, basic features and function features. The basic features, e.g. boat and place, provide simple semantic information about a word. They are organised in a hierarchy. The function features, e.g. departure_time, to_place, do not have the same structure and they are associated with actions rather than objects [11].

2.2.3 Task knowledge

The terms task and task model are often used when describing dialogue systems, but they can refer to very different phenomena. It is important to make a clear distinction between the system's task(s) and the user's task(s). A user task is non-linguistic and takes place in the real world outside the system. Models of such tasks represent the user's goals, and knowledge of how they can be achieved. Models of system tasks describe how the system's communicative acts and other tasks, e.g. database access, are carried out.

Dahlbäck [21] uses the term *dialogue-task distance* to describe the degree of connectedness between the user's non-linguistic task and the dialogue structure. For some types of dialogues it is important that the system knows what nonlinguistic task the user is performing or is planning to perform. In these cases the system can often infer the necessary information from the linguistic information. For other types of dialogue, where the dialogue-task distance is long, information about the user's task is less necessary.

Depending on dialogue-task distance different dialogue models are suitable. For advisory or problem-solving dialogues a plan-based

model, in which the user's intentions and goals are represented, might be appropriate. For these types of dialogues, there is a close connection between the task and the language, which means that it is possible to infer the non-linguistic intentions behind an utterance. In simple human-computer interaction for information retrieval, the connection between the task and the dialogue is weaker. The user's task is not necessarily reflected in the dialogue that seemingly only consists of information exchanges. This makes it harder to infer the underlying intentions if one wants to model these in the system. On the other hand that type of information is not always necessary for the system to be able to respond appropriately. For this type of interaction a dialogue grammar is sufficient [21].

User task models can vary in complexity depending on the amount of information that has to be exchanged, the structure of the task, and the negotiation necessary. A user task model can consist of a hierarchical representation of sub-tasks which captures the task structure. The structure or lack of structure is important. If a task is ill structured, the system cannot predict what kind of information the user will request and when. In a well-structured task with a predefined, at least partially ordered exchange of information, it is much easier for the system to model the interaction.

In an information-providing system a system task model can assist the system in judging if all the required parameters are present and in cases where they are not, determine what type of information to collect from the user. The use of an explicit system task model makes the system more flexible since it is easier to modify or add new tasks.

In this survey the TRAINS and CIRCUIT FIX-IT SHOP systems are the only systems that are problem-solving and have models of the user's task. The task in the TRAINS system is route planning and knowledge about the task, i.e. the planning process is represented explicitly. The plans that are developed during the interaction with the user are represented as a hierarchy of goals that are expanded into sub-goals. The hierarchy is complemented with a linear history of the planning process. For some sub-problems specialised domain servers are used [25].

In the CIRCUIT FIX-IT SHOP system the task model is prominent, and it contains information about goals, actions and states. Actions can consist of a hierarchy of sub-actions. Actions that lack sub-actions are primitive, and specify physical or sensory actions that users can perform. Goals can be accomplished by performing one or more actions. States can be either physical, representing properties or behaviours, or mental, representing beliefs about the physical state or the user's abilities. The task model can be used to answer questions about the physical state or determine what action to recommend the user to perform [60].

The other systems are all some sort of information-providing system providing information retrieved from databases. In the LINLIN, VERBMOBIL and WAXHOLM systems the system task knowledge is integrated with other knowledge sources and models, and lies implicit in the system. The GALAXYII, and RAILTEL systems have a frame-based specification form that is more or less utilised as a system task model. In the GALAXYII system there is no explicit task model but a frame representation is used to see if all the required slots are filled and, in case some are missing, ask the user for a clarification [58]. In a very similar way task rules in the grammar of the RAILTEL system are connected to the semantic frame representation of the provided information and sub-dialogues are initiated if information is missing in the frame [8].

The SUNDIAL system utilises a more sophisticated system task model that represents the task structure and keeps track of the status of the task. This often means deciding whether the user has given enough information and if not, what has to be further provided. Another role is to handle situations that arise when the provided information is incorrect. In this case the task model is used to relax the parameters instead of returning a negative answer. For example, if the user has requested a flight at noon and none is available, the system tries to find one in the morning instead [47].

2.2.4 Knowledge of the user

User models represent the user's goals and plans, capabilities, attitudes, and knowledge or beliefs. They vary in complexity, ranging from user stereotypes [53] to complete models of the user's knowledge, intentions, attitudes, etc. [42].

Depending on what kind of information a user model contains it can be used for various purposes. If a user model represents what the user knows about the domain, the system can adapt its answers so that they are informative and easily understandable. User models can also be utilised for dialogue control.

Kass & Finin [42] discuss when user models can be of great assistance and when they can be left out of a system. In simple question-answering systems no user model is necessary. In co-operative question answering systems user models can be more beneficial. For such systems a simple model of the user's goal can be used but a generic model suffices. Co-operative consultation systems on the other hand need an extensive user model.

The TRAINS system is an example of a co-operative system which uses much elaborated models of the and itself. The user model and self model represent the user's and system's mental state and contain information about their beliefs and proposals about the domain. The beliefs can be private to either the system or the user, or they can be shared by both. Mutual beliefs include aspects of the domain plans that are considered to be agreed on by both system and user. An example of private beliefs held by the user are the domain plans that have been proposed but not acknowledged. The system's private beliefs consist of the domain plans the Domain Planner has derived but have not been presented to the user [64].

The user model in the CIRCUIT FIX-IT SHOP system is a collection of axioms representing the user's knowledge about the state, as it is known by the system. These axioms are used by the system when it tries to prove a theorem and generate missing axioms, i.e. when it

decides which sub-actions have to be performed in order to complete a task. For example, if the system knows that the user knows how to perform a sub-task, it can give a higher order instruction instead of explaining and guiding the user through the sub-task [60].

2.2.5 Knowledge sources and dialogue types

To summarise the discussion of the different types of knowledge, information-providing systems and problem-solving systems are contrasted in this section.

Information-providing systems

In information-providing systems the dialogue and the information exchanged through the dialogue are the central features of the system. This is reflected by the fact that the dialogue model and dialogue history are often prominent in such systems.

Information-providing systems are in general less focused on the user's task than problem-solving systems. This is reflected in the lack of user task models. Some systems instead have explicit system task models that represent the information-seeking and information-providing tasks the system. In information-providing systems user models are not necessary, since the execution of the system's tasks is highly independent of the user's knowledge.

The type of domain knowledge needed in information-providing systems also differs from problem-solving systems. If the domain and task are fairly simple, complex domain knowledge sources might be unnecessary when instead a simple conceptual model would suffice.

Problem-solving systems

In problem-solving systems the dialogue between the user and the system is just a means of solving a problem. To identify and co-operate to achieve the user's goals is the primary concern of the system. The focus on the tasks and goals of the user is reflected in the the dialogue model and the dialogue history, which in most cases are intention-based.

Another consequence of the focus on the task is that the user task model is essential, and sometimes takes on some of the responsibilities held by the dialogue model in information-providing systems. For example, the user task model can be used instead of the dialogue model to decide how to deal with sub-goals that are not accomplished.

When a dialogue system is working jointly with the user on a task, it has to build a model of the domain or world that is being altered by actions performed by the user during the dialogue. The domain knowledge sources in problem-solving systems therefore have to be dynamic and in most cases more elaborate than the domain knowledge sources found in information-providing systems. The system also has to keep track of the user's changing knowledge about the domain, thus making a user model a necessity.

2.3 Relations between knowledge sources

To be able to respond properly to a user request in an information providing system, the system has to have knowledge about both the domain and the task. Thus, it can be very tempting to integrate this kind of knowledge in the dialogue model. The task and domain knowledge can also be mixed since the performance of a task is often domain-dependent.

Dialogue systems for information retrieval often lack an explicit system task model, in these systems the representation of task knowledge frequently becomes a part of the dialogue model. This works well in simple domains where the system only performs one or a few similar tasks but if a system is to manage several different tasks explicit task models are preferred.

A typical example of the integration of domain and task knowledge is the use of semantic frames which are part of many information-providing dialogue systems. They represent the relevant domain concepts, and are sometimes coupled to rules for interpretation of domain concepts and rules to fill in default values in a frame. Besides the role of conceptual models, the semantic frames often serve as system task models since they are used to describe what information has to be gathered by the system before a database access. This multiple use of semantic frames is useful as long as the system task is rather simple and well-structured. If the task becomes more complex, separate system task models might be needed.

The relation between user models and dialogue histories has been debated. Opinions range from user models and dialogue histories being completely separate to being two sides of the same coin [43, 57, 14]. The differences in opinion seems to a great extent to depend on how the two types of models are defined. Some researchers focus on the user's goal and compare it to the intentional level of dialogue histories, others look at the user's beliefs and knowledge and compare it to the attentional information in dialogue histories.

2.4 Summary

This chapter presented a survey of eight information-providing or problem-solving dialogue systems with the focus on the knowledge sources and models they use for dialogue management. Four types of knowledge, represented by seven different knowledge sources, were identified. **Dialogue knowledge** is represented in *dialogue models*

and *dialogue histories*. **Task knowledge** can be divided in *system task models* and *user task models*. **Domain knowledge** is held by *domain knowledge sources* and *conceptual models*. Finally, **Knowledge of the user** can be represented by a *user model*. The aim with this new categorisation was to clear up the sometimes confusing terminology.

Chapter 3

Capabilities for dialogue management

Choosing natural language as a means for interaction with a system is often motivated by the ease and naturalness of natural language. However, a system needs many capabilities if the dialogue is to be perceived as natural by the user. In this chapter such capabilities related to dialogue management are presented, and how the capabilities are supported by the knowledge sources and models presented in the previous chapter are discussed.

3.1 Graceful and co-operative interaction

Hayes & Reddy [35] describe a set of skills they consider necessary to achieve graceful interaction between a human and an information-providing dialogue system.

The need for domain knowledge[1] is stressed. Two aspects of domain knowledge that should be captured are a priori knowledge about the domain and the services and a specification of how to interact with a user during the formulation of a request. The first is related to three different skills: knowledge about the type and structure of entities in the domain, ability to derive new information from the provided information, and default information. The second involves three different skills: to know what information has been provided so far, what vital information is missing, and the focus of the current conversation.

According to Hayes & Reddy a dialogue system also needs skills for dealing with questions about its capabilities, actions performed by the system, and hypothetical questions. There are also several skills related to the goals and focus of a user interacting with an information-providing dialogue system. The user always has an overall goal with his utterances while the system is assumed to have no other goal than to co-operate with the user. However, on lower levels where the user's goal is divided into sub-goals, the system may also have goals. To deal with goals and sub-goals the system must be able to detect and adopt to the user's high level goal, to generate sub-goals, and to know how to achieve a goal. Handling the focus of the conversation requires capabilities to follow shift of focus, and to resolve anaphora and ellipsis.

Finally, since requests in information-providing dialogue systems are specified by providing a set of entities, the system needs skills for identification of entities from descriptions. An attempt to map a description to an entity can have three different outcomes: a unique entity is found, the description is ambiguous and several objects are found, or the description is unsatisfiable and no entity is found.

Abella et al. [1] provide a set of dialogue principles for successful interaction with a dialogue system. These are completion, disambiguation, relaxation, confirmation, augmentation, and user confusion/error correction. Some of these overlap with Hayes & Reddy's

[1] Hayes & Reddy do not distinguish between domain and task knowledge. Using the terminology from the previous chapter some of the domain-related capabilities would be classified as task-related instead.

skills, for example disambiguation is similar to the skills for dealing with descriptions.

Completion of a request means requesting missing pieces of information from the user. Disambiguation can be needed for two reasons, if the user has said something ambiguous, or if there are too many responses from the application. Relaxation and Confirmation are two ways to deal with requests that cannot be executed due to conflicting information. Relaxation means that the properties of a request that are considered the least important are dropped and the system tries to fulfil the new, less specific, request. Confirmation can be used by a system to check if information that seems incorrect has been understood correctly, for example if a user has provided a date that is out of the range of possible dates. Augmentation is the opposite of relaxation. If a request lacks some constraints, the system should choose the best question to ask in order to resolve the ambiguity. If a user does not know how to answer, a question or gives an erroneous answer the system must be able to find another question to ask. This is called User confusion/Error correction by Abella et al. [1].

Another set of guidelines aiming at the designing of co-operative systems for information-providing dialogues, is presented by Bernsen et al. [9]. The emphasis is on co-operativity and skills that minimise miscommunications. There are also some skills for the clarification and repair meta-communication necessary to handle misscommunications, as such cannot be totally eliminated in dialogue systems due to technical limitations.

Seven aspects of the interaction are captured by the guidelines: informativeness, truth and evidence, relevance, partner assymetry, background knowledge and repair and clarification. Each aspect is represented by one or several generic or specific guidelines, which include some of Grice's maxims [31].

The guidelines overlap the skills presented by Hayes & Reddy [35] and the principles of Abella et al. [1]; for example, to provide feedback, to deal with inconsistent or vague user input, to handle misunderstandings by the system or the user, and to provide sufficient domain

knowledge and inference. However, the guidelines also cover some other aspects of the interaction, for example, communication of the commitments made by the user, to use the same formulation of a question everywhere, to communicate what the system can and cannot do, and to provide instructions on how the user should interact with the system. They also promote user-adaptive interaction; the system should differentiate between the needs of novice and expert users whenever possible.

Another aspect of the interaction between a user and a dialogue system which is not explicitly mentioned by the principles and guidelines is initiative. To allow mixed-initiative in dialogues is, however, considered a very important feature of dialogue systems by many. Initiative can be viewed in different ways (see [15] for some approaches), I use the term mixed-initiative to mean that both user and system can control the flow of the dialogue by introducing new topics and initiating clarification sub-dialogues. Skills related to mixed-initiative dialogue include how the user might answer a question from the system, for example by ignoring it or giving too much information, and how clarification sub-dialogues can be initiated.

3.2 Capabilities for dialogue management

The skills, principles and guidelines presented above, which are partially overlapping, deal with features that are considered useful in order for a dialogue system to interact with a user in a co-operative and natural way. In this section I will reformulate them in a uniform format and introduce a classification. When skills, principles and guidelines are similar, the most generic one has been chosen. The capabilities are also presented in appendix A to facilitate future references to them.

3.2.1 Handling tasks and requests

The communication between user and system always has a purpose, to achieve a task. The task can originate from the user or from the system, depending on the service type of the system. In a problem-solving system the user's task is in focus while an information-providing system concentrates on the information requests from the user and the corresponding system task. In an information providing system, a typical sub-task is to specify a parameter in a request. A request can be quite complex and consist of a number of constraints that the user has to provide. There are several capabilities useful for handling tasks and requests:

A1 To identify the task.

A2 To identify sub-tasks and know how they are related to a task.

A3 To reason about how much of a task has been achieved so far.

A4 To decide what action to take in order to achieve a task.

A5 To deal with situations in which no answer can be retrieved from the background system.

A6 To deal with situations in which the answer from the background system includes to much information.

A7 To detect and deal with hypothetical questions.

A8 To explicitly communicate a commitment made by the user in the conversation.

3.2.2 Achieving mixed-initiative dialogue

A key to achieving a natural interaction is to allow mixed-initiative dialogue in which both user and system can take the initiative. The mixed-initiative behaviour of a dialogue system depends on the following capabilities:

A9 To allow the user to over-answer questions.

A10 To allow the user to initiate clarification sub-dialogues.

A11 To allow the user to abandon the current request and pose a new request instead.

3.2.3 Handling focus and discourse

In order to achieve a natural dialogue, a system has to know what the conversation is about. This involves knowledge of what the focus of the current conversation is and other capabilities related to the discourse:

A12 To follow shifts in focus.

A13 To resolve anaphora and ellipsis.

A14 To answer questions on what has been said and done during the conversation

A15 To answer questions about the reason why an action was performed.

3.2.4 Handling domain knowledge

Other capabilities needed to achieve a natural dialogue are related to the system's knowledge about the domain. To be an intelligent conversational partner, the system has to have sufficient domain knowledge and be able to make inferences. The domain knowledge is necessary for the system to be able to handle descriptions and processing of requests:

A16 To map a description to an entity.

A17 To detect ambiguous descriptions and deal with them.

A18 To detect erroneous descriptions and deal with them.

A19 To know the type and structure of the entities in the domain.

A20 To reason about and derive new information from the information provided by the user.

A21 To deal with a user's erroneous inferences or false presuppositions

A22 To have domain-related default information.

A23 To adapt to the user's domain expertise.

A24 To know what the system can and cannot do.

3.3 Relations between capabilities and knowledge sources

The capabilities presented all depend on the use of various knowledge sources and models. For some of the capabilities knowledge from several knowledge sources or models is required while for others one knowledge source is enough. In this section I will explore how the different capabilities can be achieved. The systems presented in the previous chapter will be used as a basis for the analysis.

A1 **To identify the task.** In systems that only perform a specific task, like the RAILTEL and WAXHOLM systems, this capability only requires a dialogue model that holds the information on how the system should behave in order to cooperate with the user to achieve the task at hand. In systems where the task is more complex or several tasks must be accomplished, explicit models of the system's or the user's tasks are used, as is the case in, for example, the SUNDIAL and the TRAINS systems, which utilise system task models and user task models respectively.

A2 **To identify sub-tasks and know how they are related to a task.** For this capability either a dialogue model, a system task model, or a user task model is needed. For the problem-solving systems, TRAINS and CIRCUIT FIX-IT SHOP, the user task model explicitly represents how sub-tasks are related to the user's task. In CIRCUIT FIX-IT SHOP a task that is not accomplished is represented by a theorem that is not yet proven, and missing axioms correspond to sub-tasks. In many of the information-providing systems, for example RAILTEL, SUNDIAL and GALAXYII, frames are used as system task models which implicitly model how sub-tasks are related to the system's task, for example, how specification of a departure location is related to the overall task of providing trip information. In information-providing systems where no explicit system task model is used, e.g. LINLIN, the relation between a sub-task and a task is represented by the dialogue model, for example by a sub-dialogue specification.

A3 **To reason about how much of a task has been achieved so far.** To accomplish this capability either a dialogue history, a system task model or a user task model can be used. The problem-solving systems, e.g. TRAINS and CIRCUIT FIX-IT SHOP, utilise user task models, while the information-providing systems that have system task models, e.g. RAILTEL and SUNDIAL, use these. Information-providing systems that lack explicit system task models, e.g. LINLIN, use a dialogue history instead, which represents what has been accomplished in the dialogue so far.

A4 **To decide what action to take in order to achieve a task.** In simple cases, e.g. systems such as LINLIN that only perform one task, a dialogue model is sufficient for this purpose. For systems that deal with a complex task or more than one task, explicit system task models or user task models are used together with a dialogue model, as in GALAXYII, SUNDIAL, TRAINS and CIRCUIT FIX-IT SHOP.

A5 **To deal with situations in which no answer can be retrieved from the background system.** This capability is primarily needed for information-providing systems where fully specified requests are used to retrieve an answer from some background system. A way to deal with this type of situation is to relax some of the constraints in the request. This approach is taken in, for example, SUNDIAL. Another approach is to present the problem to the user and let her deal with it. This requires a dialogue model that can handle this type of situation.

A6 To deal with situations in which the answer from the background system includes too much information. To achieve this capability knowledge from a domain knowledge source or conceptual model can be used to narrow the set of possible answers. The dialogue model can also provide information on how to correct the situation, primarily by entering a sub-dialogue. For instance, RAILTEL has three different dialogue grammar rules to deal with the latter type of situation: If there are less than 3 answers to a request, all information is presented; if there are 3 to 10 answers, partial information is presented and more specific information is asked for in order to narrow down the set; if there are more than 10 answers, more specific information is requested from the user [8].

A7 To detect and deal with hypothetical questions. None of the systems in the survey deals with hypothetical questions. To be able to do so the system should be able to differentiate between the task at hand and the new task that the hypothetical question involves. This requires a sophisticated dialogue model, and a dialogue history that records the current task which the system can switch back to later. A system task model or user task model might also be helpful to separate hypothetical from ordinary requests.

A8 To explicitly communicate a commitment made by the user in the conversation. This capability requires a dialogue model that can confirm the commitments made, and a system task model, user task model, or dialogue history that serve as a source of this type of information. For example, when the user has specified a trip, the provided constraints can be recorded in a task model or the dialogue history. Before accessing the background system the system can then communicate these commitments to the user by means of one of these models and the dialogue model.

A9 **To allow the user to over-answer questions.** Over-answering a question from the system means that the user provides the requested information but also takes the initiative and provides new information. To handle this requires a flexible dialogue model which allows the user several ways to respond to a system utterance. A dialogue history, system task model, or user task model is also needed for storage of the provided information. The LINLIN system accomplish this capability with the use of the dialogue model and dialogue history, while, for example, the RAILTEL system utilises the dialogue model and system task model.

A10 **To allow the user to initiate clarification sub-dialogues.** This capability requires both a flexible dialogue model and a dialogue history. If the user is to be able to postpone the answer to a question from the system, the system must remember previous states of the dialogue so that it can pick them up later in the interaction.

A11 **To allow the user to abandon the current request and pose a new request instead.** This is the third capability related to mixed-initiative dialogue, and as previously (A9-A10) this also requires a flexible dialogue model. The user must be allowed to shift topic and sometimes ignore a direct question from the system.

A12 **To follow shifts in focus.** Shifts in focus can be detected by the use of a dialogue model. In many systems, for example CIRCUIT FIX-IT SHOP, VERBMOBIL and SUNDIAL, the dialogue model supports the computation of expected types of responses, which can be used to determine whether a shift in focus has occurred. To know what the previous and current foci are also requires a dialogue history that models the attentional state of the dialogue. Such dialogue histories can be found in the GALAXYII and RAILTEL systems, for example.

A13 To resolve anaphora and ellipsis. To resolve anaphora and ellipsis a dialogue history that models the entities that have been talked about is required in order to find the possible referents. For example, a sequence of previous frames as in the GALAXYII and WAXHOLM systems, or a more structured representation like the trees in the SUNDIAL, RAILTEL and LINLIN systems, can be used for this purpose.

A14 To answer questions on what has been said and done during the conversation. For this capability a dialogue history that records the previous states of the dialogue is essential together with a dialogue model that can handle this type of question. User task models of the type in the CIRCUIT FIX-IT SHOP system can also provide information on what has been done.

A15 To answer questions about the reason why an action was performed. If the system is to be able to answer why an action was performed, it has to have a sophisticated model of the tasks that include why they were performed. User task models of the type found in the TRAINS and CIRCUIT FIX-IT SHOP systems could be used for this purpose. Of course, the dialogue model must also know how to deal with this type of question.

A16 To map a description to an entity. Since background and application systems often contain quantitative and precise data, vague qualitative terms have to be transformed to precise entities, for example the expression "this evening" can be transformed to a precise date and time interval. This can be done with a domain knowledge source, for example by the use of interpretative rules like those in the RAILTEL system.

A17 To detect ambiguous descriptions and deal with them. Descriptions can be ambiguous, meaning that there are several matching entities. Domain knowledge sources or conceptual model can be used both to decide if descriptions are ambiguous and to determine how the situation should be handled, for example by providing alternatives or asking for clarifying information. The latter strategy, of course, has to be supported by the dialogue model.

A18 To detect erroneous descriptions and deal with them. Descriptions can also be erroneous if they hold faulty or inconsistent information. In this case a domain knowledge source or conceptual model can be used to find the problem and sometimes suggest other possible interpretations that can be presented to and evaluated by the user.

A19 To know the type and structure of the entities in the domain. A conceptual model is suitable for this. In the CIRCUIT FIX-IT SHOP system a specific model that is separated from the rest of the task and domain knowledge is used for this purpose. A domain knowledge source can also hold this type of information.

A20 To reason about and derive new information from the information provided by the user. This capability can be achieved by utilising a domain knowledge source or a conceptual model, given that they are capable of reasoning about entities and concepts. For example, in the VERBMOBIL system the conceptual hierarchy of temporal entities is used to reason about and derive available times for scheduling.

A21 To deal with a user's erroneous inferences or false presuppositions. This capability requires a domain knowledge source that can reason about information provided by the user and spot erroneous inferences and false presuppositions, as well as a dialogue model that can give appropriate responses in these situations. A user model of the type found in TRAINS that represents the user's beliefs can also be useful in detecting problems of this type.

A22 To have domain-related default information. The domain knowledge sources can incorporate reasoning mechanisms to provide default information. Default values can also be incorporated in a system task model. For example, in RAILTEL default value rules that are linked to the semantic frame that represents a request are used for this purpose.

A23 To adapt to the user's domain expertise. In the CIRCUIT FIX-IT SHOP and TRAINS systems the user model holds the information needed to achieve this ability. In TRAINS the user model represents the beliefs held by the user. This information is utilised to decide what new information should be brought to the user's attention.

A24 To know what the system can and cannot do. For this capability the system must have meta-knowledge about the tasks and domain it covers. This knowledge could be part of, or be derivable from, a domain knowledge source.

An overview of the relations between knowledge sources and capabilities is given in table 3.1. The table is also presented in appendix B where the capabilities are written out. Knowledge sources that are Required to achieve a specific capability are marked by an R and L means at Least one of the knowledge sources is required.

Table 3.1: An overview of the relations between capabilities and knowledge sources and models. R stands for required and L for at Least one of.

Model/ Capability	Dial Mod	Dial Hist	Sys Task Mod	User Task Mod	Dom Mod	Conc Mod	User Mod
Task and requests							
A1	L		L	L			
A2	L		L	L			
A3		L	L	L			
A4		L	L	L			
A5	L		L		L	L	
A6	L				L	L	
A7	R	R					
A8	R	L	L	L			
Mixed-initiative dialogue							
A9	R	L	L	L			
A10	R	R					
A11	R						
Focus and discourse							
A12	R	R					
A13		R					
A14	R	R	L	L			
A15	R	L		L			
Domain knowledge							
A16					R		
A17	R				L	L	
A18	R				L	L	
A19					L	L	
A20					L	L	
A21	R	L		L	L		
A22			L		L		
A23							R
A24					L	L	

3.4 Implications for design of dialogue systems

The design of a dialogue system should be based on an analysis of the system's intended functionality and behaviour. This type of characterisation can be based on studies of corpora or other empirical studies. The requirements can be expressed in terms of the capabilities described in this chapter. Given a set of desirable capabilities, the minimum set of required knowledge sources and models can be derived with the help of table 3.1.

It is, however, important to realise that the set of models that is most suitable for some capabilities might not be sufficient to support implementation of other capabilities. The decision as to which models to use in a dialogue system must therefore be guided by the situation the system will be used in, but it is important that it is an explicit choice and that the restrictions it imposes are considered.

By introducing more knowledge sources and models a more flexible dialogue system that can more easily be extended with new capabilities may be designed. It may also facilitate modularisation and clarity of the knowledge sources and models' responsibilities.

3.5 Summary

In this chapter a number of capabilities considered useful for achieving natural and graceful interaction were presented and related to the knowledge sources and models from the previous chapter. This information can be used to support design of dialogue systems as it forces the developer to make explicit choices of which capabilities to include in the system, and to consider the limitations the design of knowledge sources and models impose on the system.

Chapter 4

Capabilities for information-providing dialogues

In order to examine what capabilities an information-providing dialogue system should have a corpus study was conducted. A set of dialogues was analysed in terms of the capabilities required to handle the phenomena occurring in the dialogues.

4.1 Corpus description

The domain for the corpus study was information about local public transportation. This domain is similar to the much studied domains of air and train journeys but differs in the complexity of the geographical domain. Users' natural way of expressing departure and arrival locations rarely coincide with the official names of the bus-stops. These locations are instead described using street or area

names, locative expressions like "close to the library", or by pointing and clicking on a map. The temporal domain is, however, very similar with expressions like "tomorrow morning" or "around eight".

As a foundation for the study, a corpus of 43 dialogues between bus travellers and personnel at the local bus traffic company was collected. Of these, 5 dialogues were removed from the corpus since they were outside the scope of the system to be. The remaining 38 dialogues included requests for information about bus and train journeys, bus routes, bus-stops, price, and lost items.

A problem when using human-human dialogues for dialogue-system development is that the person who provides the services in the dialogues does not have the same performance characteristics as a computer, and that communication between humans differs from communication between a user and a dialogue system [40]. A method called *distillation* [44] was used to solve this problem. The purpose of distillation is to rewrite human-human dialogues to more resemble human-computer interaction. The distillation is based on a set of guidelines developed with respect to the linguistic, functional and ethical properties of the desired system. The result of the distillation was a corpus of dialogues, more resembling human-computer interaction[1].

Each exchange between the 'user' and the 'system' was analysed with focus on the capabilities the system needs to interpret and respond to user utterances. The identified capabilities are presented below together with the dialogue excerpts[2] that motivate them. The capabilities are also discussed in terms of their knowledge requirements.

[1]The method of distillation has been developed and refined over the last two years. The distillation of the dialogues in the corpus used here was based on a first draft of the guidelines, which were very general [19]. More precise guidelines have been developed since then. However, since later developments have mostly focused on the linguistic and ethical properties but not as much on the functional, it should not affect the quality of the distilled dialogues used in this study since it only considers the functional properties of the dialogues.

[2]The excerpts have been translated from Swedish to English. The ungrammatical English reflects the wording of the original utterances. The English translation is shown together with the Swedish original.

4.2 Capabilities

The required capabilities found in the corpus has been grouped under the headings: tasks and requests, mixed-initiative, focus and discourse, and domain knowledge.

4.2.1 Tasks and requests

The people who answered the questions in the original corpus of human-human dialogues provide information about local public transportation, but also handle complaints, give price information and help people find lost items. The system is restricted to the tasks of providing trip information, route information and providing information on bus-stops, buses and geographical objects such as landmarks and areas. Below are some dialogue excerpts that motivate this set of tasks.

U2:	I would like to ask about line number forty seven does it stop sometime at the north gate does it have a bus stop there	ja ja skulle vilja fråga om en linje nummer fyrtiosju stannar den någon gång vid norra porten också har den någon hållplats där
S3:	no	nej

U2:	hi my name is Sandra I wonder bus to Askeby on weekdays Monday to Thursday	ja hejsan Sandra heter ja ja tänkte höra buss mot Askeby (.) på öh vardagkvällar måndag till torsdag

U8:	eh it only leaves from the bus terminus or it stops somewhere else on the way	eh den går bara från bussterminalen eller stannar den någonstans på vägen
S9:	bus four hundred fifty nine stops at Södertull and Hageby	buss fyrahundrafemtinie stannar vid Södertull å Hageby

Since the dialogues introduce several more or less complex tasks the following capabilities are required:

A1. To identify the task. Since there are several different tasks in the domain, the system must be able to differentiate between them when the user makes a request to be able to respond properly.

A2. To have knowledge of how sub-tasks are related to a task. This is exemplified in the dialogue excerpt below. The system must recognise that the user is interested in trip information and it must also know that to complete this task, the sub-tasks of specifying departure location, arrival location and time have to be performed.

U2:	hi I wonder when the six-twenty leaves now from Östersta	ja hejsan ja skulle kolla när sex-tjugo går nu från Östersta
S3:	where do you want to go?	vart vill du åka?
U4:	to to Linköping	mot mot Linköping

A3. To reason about how much of a task has been achieved so far. In order to decide which sub-task it should address the system needs to know which other sub-tasks have already been performed. In the excerpt above the system must be able to determine that information about the departure and time has already been provided in order to be able to decide whether to ask for the arrival location.

A4. To know what action to take in order to achieve a task. In the example above this means that the system needs to know how to collect information about the arrival location.

Since the primary task of the system is to answer requests for information, some of the capabilities are related to handling the request responses retrieved from the background system.

A5. To deal with situations in which no answer can be retrieved from the background system. The need for this capability is exemplified by the dialogue below.

U2:	I want wonder if you have any er buses or like express buses that go from Linlöping to Vadstena on Sunday	ja ville undra om ni hade några öh bussar eller typ expressbussar som åkte från Linköping till Vadstena på sönda
S3:	there are no buses from Linköping to Vadstena on Sundays	det finns inga bussar från Linköping till Vadstena på söndagar

4.2.2 Mixed-initiative dialogue

A prominent feature of the dialogues is that they are mixed-initiative, i.e. both the user and the system can take the initiative at every point in the dialogue, which implies that the system must be able to perform a mixed-initiative dialogue with the user. In more concrete terms this requires the following capabilities:

A9. To allow the user to over-answer questions. See for example the dialogue excerpt below where the user not only responds by answering with the requested departure information, but also takes the initiative and provides an arrival location.

U2:	hi I would like to know if there are or what bus that leaves at five or some time also around five	ja hej ja skulle vilja höra om de går eller va ä de för buss som går vid fem ti typ nån tid där också ungefär vid fem
S3:	from where do you want to go	varifrån vill du åka
U4:	from Norrköping to Linköping	från Norrköping till Linköping

4.2.3 Focus and discourse

The 'users' in the corpus frequently refer to items discussed previously in the dialogue. From this it follows that the system should have the following capability:

A14. To resolve anaphora and ellipsis. Consider the dialogue:

S5:	bus five hundred forty leaves from Skeda at a quarter to oh-six-forty five and arrives in Rimforsa at oh-seven-oh-five	buss femhundraförti går från Skeda klockan kvart i noll-sex-förtifem och är framme i Rimforsa klockan noll-sju-noll-fem
U6:	oh that's too early	ah de ä för tidigt

In the dialogue the utterance "oh that's too early" ("ah de ä för tidigt") must be interpreted as "oh oh-seven-oh-five is too early", which means that the anaphoric expression "that" ("de") must be mapped to the previously mentioned temporal expression, "oh-seven-oh-five" ("noll-sju-noll-fem").

The user also talks about different objects within the same dialogue, which means that the system should be able to:

A13. To follow shifts in focus. This is exemplified below where the user first talks about an express bus and then shifts focus to another bus.

U2:	hi I wonder when the next express bus from Borensberg to Linköping leaves	ja hejsan ja undrar när nästa (.) expressbuss (.) från Borensberg till Linköping går
S3:	the next one leaves at thirteen forty seven	ja nästa går tretton och fyrtisju
U4:	uhm and then the five-twenty when does it leave	jaha å sen fem-tjugon när går den

4.2.4 Domain knowledge

A recurrent phenomenon in the corpus is that users describe objects and entities using vague terms. Since the background system only handles precise entities, it has to provide some capabilities related to descriptions:

A16. To map descriptions to entities. An example of this can be seen in temporal expressions that have to be mapped to precise time intervals, like "evenings". Sometimes the descriptions are problematic to handle. A description can be too vague, for example an arrival location that can be mapped to a very large set of bus-stops.

U2:	hi I wonder bus two hundred and one to Vidingsjö at about half past one from the Travel Centre	ja hejsan ja undrar buss tvåhundraett mot Vidingjö vid halv två ungefär från resecentrum
S3:	there are many bus-stops in Vidingsjö, can you be more precise and give a landmark or a street	det finns för många hållplatser i Vidingsjö, kan du vara mer precis åh ge ett landmärke eller en gata?
U4:	uhh then I'm going to something called Tuvgatan it's near the last stop	öhh sen ska jag till något som heter Tuvgatan det ligger vid sista stationen

This can be dealt with by asking the user for a specification, preferable by telling the user what kind of information would be helpful.

A17. To detect and deal with ambiguous descriptions. Ambiguous descriptions are those that cannot be directly mapped onto a precise entity, for example "at nine" ("klockan nie") in the dialogue below.

U1:	hi I-I want to know a suitable bus from Linköping to Söderköping on a Friday I should be there at nine	hejsan a-ja skulle vilja ha reda på en bra buss från Linköping till Söderköping en freda ja ska va där klockan nie
S2:	do you want to go in the morning or the evening	vill du åka på morgonen eller kvällen?
U3:	morning	morgonen

In such situations the system can resolve the ambiguity by providing alternatives to the user and ask her to choose one. Handling an ambiguity involves reasoning about what information is needed to precisely specify an entity and/or present some alternatives that the user can choose from. In the example, the system needs to know that an unambiguous time can be specified as a timepoint and a part of

day specification. The system can then provide two alternatives for the user to choose from. The same type of knowledge is needed to deal with descriptions that are very vague in order to help the user specify the intended entity.

To achieve natural interaction between the user and the system, the system must be able to act intelligently with regard to the domain and the application, which means that the system must be able to reason about the domain. Two specific cases of domain reasoning that were found in the corpus were the use of default values and the derivation of new information from old.

A22. To provide domain-specific default information. In some situations users leave out information that they assume is specified by the context. This type of information should therefore be inserted by the system utilising knowledge about the domain. In the dialogues it is, for instance, often the case that the user does not provide information about the date but assumes it to be the current day, see below.

U2:	hi I'm Anna Nilsson and I want to go by bus from Ryd's Centre to the Travel Centre in Linköping I should be at the Travel Centre before fourteen thirty five because we are going to the long distance buses	jaa hej Anna Nilsson heter jag och jag vill åka buss från Ryds centrum till resecentrum i Linköping å ja ska va på rececentrum innan fjorton å trettifem fö vi ska till långfärdsbussarna

A20. To derive new information from the information provided by the user. The most common example of this is that the system has to figure out what the suitable bus-stops are given a description of the arrival or departure location in a trip specification. For example:

U2:	hi I want to know how I can travel er from the hospital down to IKEA in Linköping	ja hejsan ja skulle vilja veta hur jag tar mej eh från sjukhuset ner till IKEA i Linköping
S3:	bus two hundred and twelve leaves from the US machine centre thirteen fifty seven then you change at the Travel Centre to two hundred and fifteen	buss tvåhundratolv går från US maskincentral tretton å femtisju sedan byter du ner på rececentrum till tvåhundra femton

These capabilities require a representation of the domain entities, and methods to reason about their properties and relations.

Finally, to avoid frustrated users it is important:

A24. To know what the system can and cannot do. In the distilled dialogues the answers provided by the personnel have in some cases been exchanged for a message on the system's limited capabilities.

U4:	uhh travel card how much is it	ähh månadskort vad kostar de
S5:	I cannot give information on prices, you can call 020-211010 for information on prices	jag kan inte ge information om priser, du kan ringa 020-211010 för prisupplysningar

The system, thus, needs meta-knowledge about its capabilities and limitations. To make the system more useful it would also be appropriate to direct the user to another information source in cases when the system does not have the answer, as in the example above. This, of course, means that the system must have access to this kind of 'hint' information.

4.3 Summary

To summarise, the corpus study presented in this chapter indicates that the following capabilities are required by an information-providing dialogue system: A1-A6, A9, A12-14, A16-A17, A20, and A24. The capabilities are listed in figure 4.1.

	Desirable capabilities
A1	To identify the task.
A2	To identify sub-tasks and know how they are related to a task.
A3	To reason about how much of a task has been achieved so far.
A4	To decide what action to take in order to achieve a task.
A5	To deal with situations in which no answer can be retrieved from the background system.
A6	To deal with situations in which the answer from the background system includes too much information.
A9	To allow the user to over-answer questions.
A12	To follow shifts in focus.
A13	To resolve anaphora and ellipsis.
A16	To map a description to an entity.
A17	To detect ambiguous descriptions and deal with them.
A20	To reason about and derive new information from the information provided by the user.
A22	To have domain-related default information.
A24	To know what the system can and cannot do.

Figure 4.1: The capabilities required by an information-providing dialogue system as found in the corpus study.

The capabilities in the table should, however, not be seen as a complete list as some useful capabilities might not have been present in the dialogues in the corpus. It seems, for example, reasonable to believe that whenever capabilities A16 and A17 are required, capability

A18 will also be needed. The same holds for capabilities A10 and A11, which are closely related to capability A9. The capabilities discovered in the corpus serve, however, as a minimal set of capabilities that should be supported by information-providing dialogue systems.

Chapter 5

Domain knowledge management in the MALIN framework

To easily and rapidly build dialogue systems for new domains and new tasks, dialogue system frameworks are highly beneficial. This assumption rests on *The Domain-independence Hypothesis* proposed by Allen et al. [6, p. 1] which states that:

> Within the genre of practical dialogue, the bulk of the complexity in the language interpretation and dialogue management is independent of the task being performed.

The hypothesis is supported by their work and implies that frameworks can be developed and used to create new applications by addition of task and domain knowledge that is specific for the application.

In this chapter two such dialogue system frameworks, LINLIN and MALIN are presented. The MALIN framework is an extension of the LINLIN framework, including a new separate module for domain knowledge management, a Domain Knowledge Manager, which is motivated by the results from the corpus study presented in chapter 4.

5.1 The LINLIN framework

In chapter 4 LINLIN was one of the dialogue systems presented and discussed. LINLIN is, however, not just one system but rather a framework for the development of information-providing dialogue systems. In this section the framework is discussed in more detail and analysed in terms of the capabilities found in the corpus presented in chapter 4 which are required to deal with complex information-providing dialogues.

5.1.1 The CARS system

The LINLIN framework has been customised for two different applications, TRAVEL and CARS. The cars system is currently running and can be accessed through the Internet and will therefore be used for illustration. It is a dialogue system that can provide information on second-hand cars. An example dialogue with the CARS application is shown in figure 5.1

As seen in the dialogue, users can pose complex requests for sets of objects that match certain criteria (see U1), or different properties, like price, speed, and size, for an object or a set of objects (e.g. U3 and U5). The dialogue is connected, in U5 the user can refer back to the set presented in S2. U7 illustrates how the user can shift the focus during the interaction.

U1: show all volkswagen from 1993

S2:

Manufacturer	Model	Year
Volkswagen	Golf	1993
Volkswagen	Vento	1993
Volkswagen	Passat	1993
Volkswagen	Passat Variant	1993
Volkswagen	Caravelle	1993

U3: which is the cheapest?

S4:

Manufacturer	Model	Year	Price
Volkswagen	Golf	1993	67700

U5: and the most expensive

S6:

Manufacturer	Model	Year	Price
Volkswagen	Caravelle	1993	146000

U7: are there any less expensive volvos?

S8:

Manufacturer	Model	Year	Price
Volvo	850 GL. GLE	1993	128700
Volvo	850 GL. GLE Kombi	1993	136700
Volvo	940 GL	1993	113400

Figure 5.1: A dialogue between a user and the CARS application that is based on the LINLIN dialogue system framework.

5.1.2 Architecture

The LINLIN framework has a modular architecture with an Interpreter, a Dialogue Manager and a Generator (see figure 5.2).

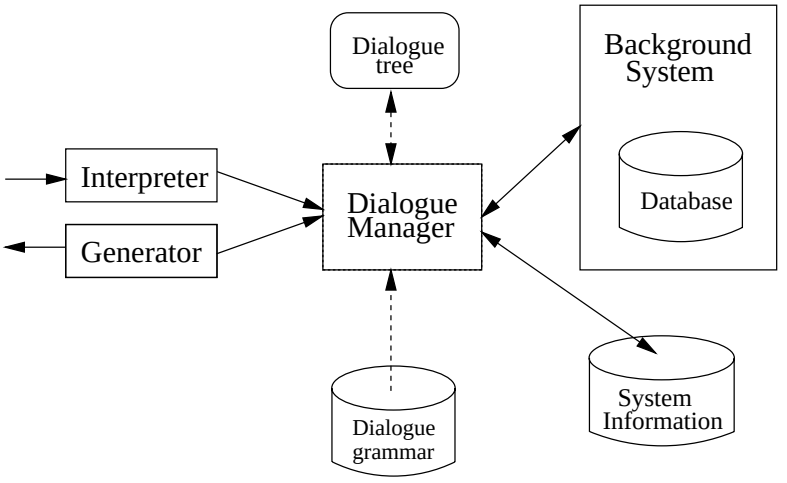

Figure 5.2: The architecture of the LINLIN framework for information-providing dialogue systems.

The role of the *Interpreter* is to analyse the user input and deliver a meaning representation that can be used by the Dialogue Manager. It does this utilising a unification-based parser [63].

The *Dialogue Manager's* primary tasks are to handle the dialogue with the user and to keep track of the interaction. Managing the interaction with the user involves deciding whether a user request is clear enough to access the background system, or if it is not, to initiate a clarification sub-dialogue. The type of background system used is databases where information can be extracted by posing questions using, for example, SQL. Accessing the background system thus involves a transformation of a request to an SQL-question and the transformation of the result from the database to a format suitable

for the Dialogue Manager. The system also utilises a database with system information to answer system-related requests.

The *Generator* is responsible for the realisation and presentation of questions and answers specified by the Dialogue Manager. In the LINLIN framework the Dialogue Manager has control of the system and decides when and which of the different modules should perform a specific task [38].

5.1.3 Dialogue management

In the LINLIN framework dialogue management primarily means to control the flow of the dialogue by deciding how the system should respond to user utterances. As a basis for this, the Dialogue Manager in the LINLIN framework utilises a dialogue model and a dialogue history.

In the dialogue model used in the LINLIN framework the dialogue is structured in terms of discourse segments, and a discourse segment in terms of moves and embedded segments. An initiative-response (IR) structure determines the compound discourse segments, where an initiative opens the IR-segment and the response closes the IR-segment [20]. The discourse segments are classified by general speech act categories, such as question (Q) and answer (A) [39], rather than specialised (cf. [33]), or domain-related [5] ones. The action for the Dialogue Manager to carry out, as modelled in a dialogue grammar, depends on how domain entities are specified and their relation to other entities in the domain and the dialogue history.

Two important concepts have been identified: termed Objects and Properties where Objects models the set of objects in the database and Properties denotes a complex predicate ascribed to this set. The parameters in Objects and Properties are application-dependent. Markers are also utilised for various purposes [41], for example to mark yes/no questions. Structures that represent information about objects,

properties and markers are termed OPMs. Figure 5.3 shows an example OPM which represents the request "How fast is a Volvo 850?".

$$\left[\begin{array}{ll} Obj: & \left[\begin{array}{ll} Manufacturer: & Volvo \\ Model: & 850 \\ Year: & \end{array}\right] \\ Prop: & \left[\begin{array}{ll} Aspect: & speed \\ Value: & \end{array}\right] \end{array}\right]$$

Figure 5.3: An OPM for the question "How fast is a Volvo 850?"

In the LINLIN framework there is only one dialogue history, which is maintained by the Dialogue Manager. Thus, the other modules in the system have no memory of the previous interaction since this could cause conflicts. The dialogue history records focal information, that is, what has been talked about and what is being talked about at the moment. It is represented as a dialogue tree and the nodes in the dialogue tree record information in OPMs.

5.1.4 Capabilities of LINLIN

In section 5.1.1 some of the LINLIN framework's capabilities were exemplified for the CARS application. In this section the LINLIN framework will be more thoroughly examined in the light of the capabilities it is designed to support.

Tasks and requests

Since the LINLIN framework was designed to deal with rather simple tasks and requests, it only provides capabilities A1, To identify the task, and A4, To decide which action to take in order to achieve a task, of the capabilities related to tasks. To detect a task means to

differentiate between domain-related and system-related requests for information, which is performed by the Interpreter based on features in the user utterances. The co-operation needed to fulfil a task is in LINLIN captured by the dialogue model. For example, if the system needs to make a clarification in order to be able to retrieve the requested information, this is captured by the pattern **I (I R) R** in the dialogue model. The first I is the user's initiative where a request is posed to the system, the (I R) the clarification dialogue initiated by the system, and the second R the resulting response to the first initiative.

Problematic responses to requests are only handled to some extent in the LINLIN framework. Capability A6, Too many responses to a request, is dealt with by asking the user for a selection, thus primarily relying on the dialogue model and without using any domain knowledge to facilitate handling of the situation. If no answers to a request can be found in the background system, capability A5, it is stated to the user who has to decide how to proceed.

Mixed initiative dialogue

All the capabilities related to mixed-initiative dialogue are provided by the LINLIN framework. Users can over-answer questions, capability A9, since the dialogue model can incorporate the information provided by the user regardless of whether it was asked for or not. Initiating a clarification sub-dialogue, capability A10, means inserting a new initiative-response unit in the dialogue tree as a child to the IR-unit that awaits the response from the user. Once the sub-dialogue is finished, the dialogue is resumed where it was. Posing a new request and thereby abandoning the current task, capability A11, is allowed by the dialogue model since it is possible for an Initiative to never receive the corresponding Response-part in an IR-unit.

Focus and discourse

In its present form the LINLIN framework can handle both changes of focus, and anaphoric expressions and ellipsis, capabilities A12 and A13, by utilising the dialogue tree that records focal parameters (OPMs) for every Initiative and Response made by either the user or the system. Changes in focus are introduced by new Objects or Properties in the OPM structure. Anaphoric expressions refer to objects or a set of objects that are represented by the OPMs in the dialogue tree. Ellipsis can refer to properties of objects and these are also found in the dialogue tree.

Domain knowledge

The use of domain knowledge in the LINLIN framework is very limited and in the CARS system there is no explicit domain knowledge source exist. This is due to the fact that the capabilities relying on domain knowledge are not necessary in this system, there are no vague descriptions to map to entities. The parameters in the OPMs that describe objects are precise and used to retrieve sets of objects.

There is, however, a knowledge source with system information which is used to provide capability A24, to know what the system can and cannot do.

Summary

The capabilities the LINLIN framework supports in its present form are summarised in figure 5.4. The table shows that the LINLIN framework is designed to handle only a small subset of the capabilities detected in the corpus study, see figure 4.1.

Capabilities of LINLIN	
A1	To identify the task.
A4	To decide what action to take in order to achieve a task.
A6	To deal with situations in which the answer from the background system includes too much information.
A9	To allow the user to over-answer questions.
A10	To allow the user to initiate clarification subdialogues.
A11	To allow the user to abandon the current request and pose a new request instead.
A12	To follow shifts in focus.
A13	To resolve anaphora and ellipsis.
A24	To know what the system can and cannot do.

Figure 5.4: The capabilities provided by LINLIN.

5.1.5 Shortcomings of LINLIN

In this section the addition of some of the capabilities not covered by the LINLIN framework are considered, and the consequences for the knowledge needed to provide them are discussed.

Tasks and requests

In LINLIN there is only one type of simple task-related request, but in many domains the system must deal with more complex requests. Complex requests are concerned with the specification and construction of compound objects, like trips in the RAILTEL domain. The specification of such an object requires that the user provides information on a specific set of parameters, which often involves several dialogue turns. The specification is used to construct a matching object by retrieving, and sometimes integrating, knowledge from one or several domain knowledge sources and background systems. To

answer requests on a trip, the system needs to have a number of parameters specified, such as departure and arrival time and place, before it is able to access the timetables.

A complex task-related request introduces sub-tasks and the capabilities related to these (A2 and A3). Thus LINLIN should be modified in order to provide these capabilities. The table 3.1 indicate that a system-task or user-task model can be useful for this purpose.

With the introduction of the complex task-related requests and compound objects they describe follows an increased risk of problematic responses to requests. Capability A5 should thus be added and capability A6 has to be elaborated so that LINLIN can deal with these responses. Table 3.1 also shows that to deal with problematic responses knowledge about the domain and the objects and entities in the domain is useful.

Domain knowledge

The complex task-related requests and compound objects also require capabilities to deal with descriptions that have not been necessary in LINLIN. The system must be able to map descriptions to entities (A16) and to detect and deal with ambiguous and erroneous descriptions (A17 and A18). It should also be able to derive new information from that provided by the user (A20) and to provide domain-specific default information (A22). All these capabilities rely on the availability of domain knowledge and domain reasoning. Thus, LINLIN has to incorporate these types of knowledge and reasoning mechanisms.

Summary

The capabilities needed to come to term with the shortcomings of the LINLIN framework are summarised in figure 5.5.

Desirable capabilities of LINLIN	
A2	To identify sub-tasks and know how they are related to a task.
A3	To reason about how much of a task has been achieved so far.
A5	To deal with situations in which no answer can be retrieved from the background system.
A6	To deal with situations in which the answer from the background system includes too much information.
A16	To map a description to an entity.
A17	To detect ambiguous descriptions and deal with them.
A18	To detect erroneous descriptions and deal with them.
A20	To reason about and derive new information from the information provided by the user.
A22	To have domain-related default information about requests.

Figure 5.5: Desired capabilities for information-providing dialogue systems not provided by the LINLIN framework.

To provide these capabilities the LINLIN framework has to be extended. The introduction of more complex task-related requests have many consequences; it indicates that the dialogue model and dialogue history should be modified or that a task model should be introduced in order to provide the capabilities required to deal with tasks and sub-tasks, see table 3.1. More complex descriptions and problematic responses to requests also have to be dealt with, as well as descriptions and various forms of domain reasoning. All these capabilities demand that domain knowledge and reasoning are incorporated in the system.

5.2 The MALIN framework

The previous section showed that the LINLIN framework lacks some of the capabilities needed for information-providing dialogue systems. A new framework called MALIN[1] has therefore been developed based on the shortcomings presented above. In this framework much effort has been put on domain knowledge management since many of the shortcomings were capabilities that require domain knowledge.

5.2.1 Architecture

An important issue for a framework is that it should be easy to customise to new domains and that various dialogue strategies can be explored. The Dialogue Manager should only be concerned with phenomena related to the dialogue with the user. It should not be involved in the process of accessing the background system or performing domain reasoning. I therefore propose that a separate module, a Domain Knowledge Manager, devoted to access of background systems and domain knowledge sources should be introduced [27]. The Dialogue Manager is still the central controller of the interaction but it co-operates with the Domain Knowledge Manager with the aim of achieving a natural and intuitive dialogue. The architecture of the MALIN framework is presented in figure 5.6.

Similar approaches have been adopted by others. In the TRIPS dialogue shell [6], a generalisation of the TRAINS system for plan-based tasks, a Behavioural Agent separated from dialogue management is responsible for communication with the back-end system. The motivation for this separation is that dialogue management can be made more domain-independent as the domain-specific features are encapsulated by the Behavioural Agent. However, the Behavioural Agent is

[1] The acronym stands for Multi-modal Application of LINLIN, since the LINLIN framework was also modified to deal with multi-modal dialogues. This aspect of MALIN is, however, not covered here, since the focus is on modifications made with respect to the new knowledge requirements.

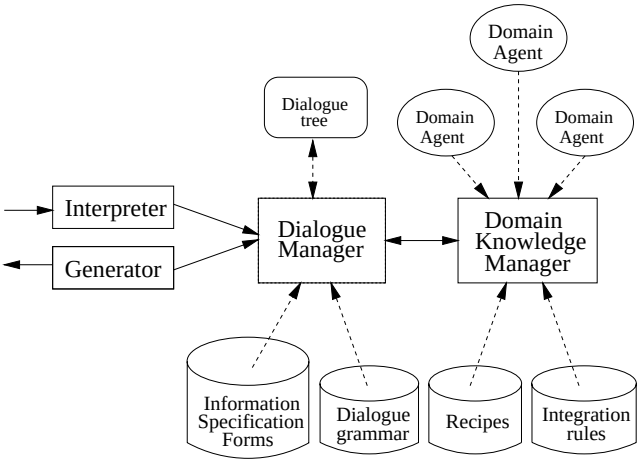

Figure 5.6: The architecture of the MALIN dialogue system framework.

able to ignore requests from the Discourse Manager and instead give information that it considers more important, thus, taking on some of the dialogue management responsibilities. The Domain Knowledge Manager differs from the Behavioral Agent in this respect, the Domain Knowledge Manager is obliged to answer a request from the Dialogue Manager and cannot take any initiatives on its own. Another similar approach is taken in the CMU Communicator system [54] where the Dialogue Manager makes use of domain- and task-specific information represented as schemas but much of the domain knowledge and reasoning has been separated from dialogue management and handled by domain agents. There seems, however, not to be any specialised module for co-ordination of the agents.

5.2.2 Dialogue management

To achieve the capabilities related to complex requests, the dialogue model and dialogue history has been complemented with system task models. Explicit system task models were chosen because they can support several tasks and facilitate modification and addition of new tasks.

The system task models hold the parameters that have to be specified before successful access of the background system can be performed. They are known as Information Specification Forms (ISFs) [19]. Just like OPMs, the ISFs are application-dependent and they are also used to record information in the dialogue tree, i.e. to inform the Dialogue Manager which parameters have to be provided by the user and which should be asked for next.

By introducing this new structure in parallel with the old OPMs, the system can follow shifts in focus and maintain the mixed-initiative character of the interaction with the user. For example, if the user has posed an incomplete complex request and the system asks for the parameters necessary to specify the request, the user might want to ask a clarifying question before giving an answer. This shift in focus can be represented in the dialogue by the introduction of an OPM. When the clarification question has been answered, the complex task represented by the ISF can be picked up again.

5.3 Domain knowledge management

Much effort have been spent on natural language understanding and dialogue management in dialogue systems. The communication with and the use of external resources containing domain knowledge and application information are in general not discussed. This is probably due to the fact that access to such resources is considered rather straightforward, but there are several capabilities related to this and some problematic situations that have to be dealt with.

Ambiguous and indeterminate knowledge have been dealt with to some extent, often by adding specialised components to the dialogue system. For example, vague temporal expressions must be resolved in dialogue systems in the ATIS and train travel domain ([23, 37]). In most cases these components extend the functionality of the interpreter and are called before a request is delivered to the Dialogue Manager.

A solution to the problem of mapping descriptions provided by the Dialogue Manager to a suitable representation for the external database has been proposed by Whittaker & Attwater [69]. A new component, the Information Manager, that interacts with the application database is introduced. The Information Manager is responsible for translation between high-level requests and a set of operations on the database. To perform its task it utilises a data model consisting of several vocabulary models, e.g. spelt out vocabulary and spoken vocabulary, and it also has knowledge of synonyms and homophones.

Similarly a separate module, the Action Manager, for communication with external resources has been proposed [52]. In this architecture the Dialogue Manager is responsible for the translation between the requests and the actions carried out by the Action Manager. When a requested action is delivered to the Action Manager, it has to decide how the action should be executed and which source to access.

Even if some of the presented architectures include a separate module for accessing the application or reasoning with domain knowledge, they deal only with one of the problems, or a monolithic application. None of the current approaches deals with all the capabilities related to domain knowledge management and the integration of several domain knowledge sources. A more sophisticated architecture is needed to handle all these issues.

5.3.1 Domain knowledge management capabilities

Of the capabilities revealed by the corpus study (figure 4.1), but not supported by LINLIN, capabilities A5-A6, A16-A18, A20 and A22, are related to domain knowledge and background system access.

Access of background systems can be problematic in two ways: no answer to the request can be produced (A5), or too many answers are found (A6). The first situation can occur if a request is inconsistent or if no object meets all the restrictions of the request. Two different approaches to dealing with this are for the system to try and fix it itself, or for the system to help the user to handle the situation. The first approach includes relaxing some of the constraints or resolving the inconsistency, both of which require reasoning about the domain. If the system fails or does not try to solve the problem itself, it can give the user as much help as possible when he or she has to deal with the problem, for example by stating the cause of the problem and suggesting how the request should be modified. MALIN supports both approaches. The second problematic situation, in which a request has resulted in too many answers from the background system, can arise from requests that are not specific enough. The solution chosen in MALIN is to use the domain knowledge and decide which constraints should be asked for in order to specify the request, thus helping the user to formulate a more specific request.

Mapping descriptions to entities (A16) is similar to retrieving answers to requests. There are also two problematic cases that have to be dealt with in this case, the descriptions can be ambiguous (A17) and correspond to several entities, or they can be unsatisfiable (A18) and not correspond to any entity. Ambiguous descriptions can be dealt with by either asking the user to choose one of the matching entities or ask the user to provide a distinguishing feature. In MALIN both approaches are used depending on how many alternatives there are. Unsatisfiable descriptions can be dealt with in three different ways: find and inform the user of faulty presuppositions that cause the description to be unsatisfiable, find and present near misses by

relaxing some of the features in the description, or inform the user of the problem giving as helpful information as possible. MALIN uses the first and third of these; if a faulty presupposition is present it is presented to the user. Other problems are also brought to the user's attention together with helpful information on how they can be corrected.

Other capabilities relying on domain knowledge include reasoning about and deriving new information from the information provided by the user (A20) and using domain-related default information about requests (A22). If a user has specified a request that is not complete, the system can fill in the empty spaces by inserting default information or by deriving it from the information that has been provided.

5.3.2 The Domain Knowledge Manager

A separate module for domain knowledge management, the Domain Knowledge Manager, was designed to supply the capabilities related to domain knowledge. The primary responsibility of the Domain Knowledge Manager is to provide domain- and application-specific information when the Dialogue Manager has produced a request. The Dialogue Manager can deliver a request to the Domain Knowledge Manager and in return expect an answer retrieved from the background systems or the domain knowledge sources. The Domain Knowledge Manager maps descriptions to entities and reasons about where and how the information should be retrieved and how information from different domain or application knowledge sources should be integrated. If the Domain Knowledge Manager encounters a problem it cannot solve by using domain knowledge, a specification of the problem and the needed clarifying information is returned to the Dialogue Manager.

Multi-agent framework

To facilitate addition, replacement, and reuse of domain knowledge sources an agent-based architecture has been chosen for the Domain Knowledge Manager. The Domain Knowledge Manager consists of several agents: the **Control Agent**, the **Recipe Agent**, the **Integration Agent**, and **Domain Agents**. The agents provide different *services*, for example to retrieve some information given some parameters, and can also request services from each other. Communication and co-operation among the agents are achieved by passing messages in ICL, the Interagent Communication Language that is a logic-based declarative language that can represent natural language expressions.

The Control Agent is a generic domain-independent agent that controls the processing of a request. For this purpose it utilises knowledge structures called *recipes*. A recipe (cf. recipe-for-action [50]) consists of a series of services from different agents, which are executed in order to construct an answer to the request.

The Recipe Agent is responsible for the construction of recipes that match the requests. The recipes are application-specific, but the agent in itself is domain-independent.

The Integration Agent is a domain-independent agent that can integrate several response alternatives into one answer, utilising *integration rules*, which contain both domain heuristic and more general principles.

The Domain Agents are responsible for appropriate access of domain knowledge sources and are able to perform sophisticated knowledge reasoning in order to retrieve the information. Thus, each domain agent provides a set of services such as storing, retrieving or constructing a specific type of information. The domain agents are in general application-specific. Some more detailed examples of domain agents are provided in chapter 6 that covers the ÖTRAF application.

The implementation of the Domain Knowledge Manager is based on the Open Agent Architecture, OAA, which is a framework for the development of multi-agent systems [46]. Some characteristics of OAA are that it is **open**, meaning that agents can be created in multiple programming languages and interface with existing legacy systems, **extensible**, as agents can be added or replaced individually at runtime, **distributed**, since agents can be spread across any network-enabled computers, **parallel**, as agents can co-operate or compete on tasks in parallel.

How the agents work together to process a request is described in the following section. To make the presentation less abstract, examples from the local public transportation domain will be used when called for.

Processing of requests

Processing of a request delivered by the Dialogue Manager, in general involves three steps. First the Domain Knowledge Manager has to decide how to treat the request, i.e. to produce one or more recipes. In most cases one recipe is enough, but sometimes the user has provided ambiguous information that cannot be resolved by the interpreter or the Dialogue Manager. In these cases several recipes are needed. The next step is to process the recipe(s). The processing must be carefully monitored and problematic descriptions or responses dealt with. Finally alternatives must be inspected and integrated into one answer that can be sent back to the Dialogue Manager.

Producing the recipes

The first step towards fulfilling a request is to find a suitable recipe that describes what information is needed and how the information should be retrieved. For this purpose the **Recipe Agent** provides a makeRecipe service. As a basis for the production of a new recipe,

a recipe library that contains recipe templates for different types of requests is utilised. The recipe agent maps the request to a suitable recipe template and instantiates it with the values from the request. Figure 5.7 shows a recipe template containing two service-calls to two different agents. Agent1 retrieves some information that is used as input to the service provided by Agent2.

Agent	*Service*	*Parameters*	*Result*
Agent1	service1	feature1, feature2, ..., feature*n*	Result1
Agent2	service2	feature*m*, Result1	Result2

Figure 5.7: An schematic view of an uninstantiated recipe template.

Since the information given by the Dialogue Manager can be ambiguous, the Recipe Agent in some cases has to find one or more unambiguous interpretations of the request. Ambiguous requests result from ambiguities that cannot be resolved by the interpreter or the Dialogue Manager during interpretation, for example names denoting several objects of different types.

The disambiguation can be approached in several ways: one is to ask the user for clarification, but this can result in a cumbersome and unnatural dialogue. Our approach is instead to generate several interpretations originating from the ambiguous information, and produce a recipe for each interpretation, and process them in parallel. The different interpretations can then be evaluated depending on the result, and subsequently be integrated into one answer.

The same approach can be used to handle alternatives in requests. For example, a user may say "I want to go to Norrköping on Friday night or Saturday morning". Since bus trips can only have one travel time, this is treated as two separate requests, and two different recipes are produced.

```
Input: Request, RecipeDB
Output: Recipes

Requests := null

if (Request.isAmbiguous()) then
   Requests := Request.disambiguateSplit()

else if (Request.containsAlternatives()) then
   Requests := Request.alternativesSplit()

else
   Requests.add(Request)

Recipes := null

for each Req in Requests do
   Recipe := Req.matchRecipeTemplate(RecipeDB)
   Recipe.instantiate(Req)
   Recipes.addRecipe(Recipe)

Recipes.connector := Requests.connector

Return Recipes
```

Figure 5.8: The algorithm used to make recipes takes a database of recipe templates and a request as input and outputs a set of recipes.

The algorithm used to make recipes is presented in figure 5.8. The disambiguateSplit() method is a domain-dependent means of splitting ambiguous requests while the alternativesSplit() is a domain-independent way of splitting requests that contain alternatives. When a request is split, it results in a set of requests that are related by a connector; typically the connector for alternatives is OR, but there might also be domain-dependent connectors that order the requests according to some criteria.

Processing of recipes

As mentioned above, the main purpose of the Domain Knowledge Manager is to collect and integrate information from various domain agents. A complex request involves access of many different domain agents that map descriptions to entities or retrieve the requested information. Which of the agents' services and in what order they should be called are specified in the recipes. The **Control Agent** supervises the process and deals with problematic descriptions or request responses, and errors.

Each of the recipes produced is processed and the results are collected. The responses from the different recipes are then integrated into one. In this process some of the problematic responses might be discarded. For example, if an ambiguous request was split into several recipes and some of these resulted in errors these can be ignored if there is some recipe that has produced an appropriate response.

After integration of the responses there may still remain problems, for example a description that could not be successfully mapped to an entity or a request for which no response or too many responses could be retrieved. For too narrow requests the Domain Knowledge Manager tries to relax them while for the other problems an explanation will be sent to the Dialogue Manager. The explanation typically includes the cause of the problem and a suggestion as to how it can be handled, for example alternatives that can be presented to the user for her to choose from or parameters that should be specified. This is illustrated by the algorithm in figure 5.9

```
Input: Request
Output: Response or Problem

Recipes := RecipeAgent.makeRecipes(Request)

Responses := null

for each Rec in Recipes do

   for each ServiceCall in Rec do
      Result := ServiceCall.execute()

      if (Result.getError() != null) then
         SendResponse(ERROR, Result.getError())

   Responses.add(Result)

Response := IntegrationAgent.integrateResponses(Responses)

if (Response.getError() == NoResponse) then
   NewRequest := Request.relaxConstraint()
   NewRequest.process()

else if (Response.getProblem() != null) then
   SendResponse(PROBLEM, Response.getError())

else
   SendResponse(SUCCESS, Response.getResult())
```

Figure 5.9: The algorithm used to process a request takes a request as input, asks the RecipeAgent to make the corresponding recipes, processes the recipes, asks the IntegrationAgent to integrate the responses and outputs a response.

Integrating the alternatives

Since ambiguous requests and requests containing alternatives result in a set of recipes which are processed independently, the results must be integrated before an answer is transferred to the Dialogue Manager. This is performed by the **Integration Agent** utilising integration rules. Some of the rules are general and domain-independent while others are heuristic and specialised for a specific domain. A simple domain-independent rule that is used to integrate alternatives states that: If all but one recipe have failed, the successful one is returned to the dialogue manager. If there are several successful recipes, more complex domain-dependent rules can be used to sort and filter the alternatives. For example, in the travel domain travel time or cost can be used to rank the trips. When there are no successful responses at all, rules are used to how errors should be reported and dealt with. How several responses can be integrated into one is illustrated by the algorithm in figure 5.10.

5.4 Summary

In this chapter the MALIN framework, which is an extension of the LINLIN framework, has been presented. In this framework a new separate module for domain knowledge management, the Domain Knowledge Manager, was introduced. The Domain Knowledge Manager supports several capabilities related to domain knwoledge and access of background systems. An advantage of the separation of dialogue management and domain knowledge management is that the Dialogue Manager become more domain-independent and focused on the dialogue features. Another advantage is that domain knowledge sources can be added, exchanged and reused more easily.

```
Input: Responses, RuleDB
Output: Response

TempRes := null

for each Res in Responses do
   if (Res.getError() == null) then
      TempRes.add(Res)

if (TempRes.length() == 1) then
   Response := TempRes

else if (TempRes.length() > 1) then
   Rule := TempRes.matchRule(RuleDB)
   Response := TempRes.apply(Rule)

else
   Rule := Responses.matchRule(RuleDB)
   Response := Responses.apply(Rule)

Return Response
```

Figure 5.10: The algorithm used to integrate responses takes a database of integration rules and a set of responses and outputs a response.

Chapter 6

Domain knowledge management in the ÖTraf system

In this chapter an instance of the MALIN framework, a dialogue system that provides information about local public transportation, the ÖTraf system, is presented. The knowledge sources and models used by the Dialogue Manager and the Domain Knowledge Manager are presented. An example dialogue is used to illustrate how the Dialogue Manager and Domain Knowledge Manager co-operate to achieve a natural dialogue with the user.

6.1 Architecture and information flow

To create a new application from the MALIN framework the knowledge sources and models used for dialogue and domain knowledge management need to be designed and implemented. For the ÖTraf system

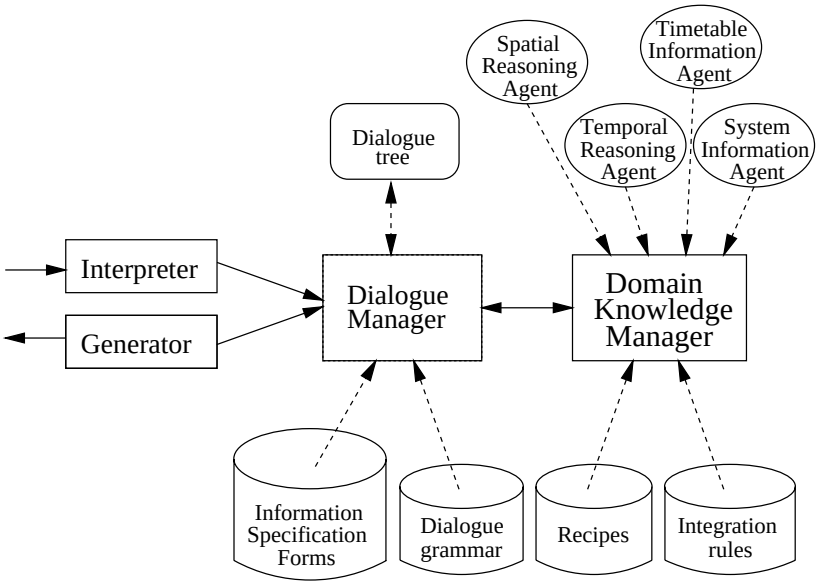

Figure 6.1: The architecture of the ÖTraf dialogue system.

an application-specific Dialogue Grammar and Information Specification Forms, ISFs, has been developed. The information structures used in the dialogue tree have also been customised to the specific domain and tasks. For domain knowledge management, application-specific recipes and integration rules have been specified. Domain agents that hold knowledge of the spatial/geographical domain, temporal domain, timetable information for the local public transportation, and system and help information have been added. All these knowledge sources and models are shown in figure 6.1 that provides an overview of the ÖTraf system architecture.

The full arrows in the figure indicate how information flows between the modules. A user utterance is parsed by the Interpreter and a feature structure containing the relevant information in the utterance is passed on to the Dialogue Manager. The Dialogue Manager then

takes the given information and processes it in the dialogue context. This means that the new information may be incorporated in a previous partially specified request or that it results in a new request. As described in chapter 5, simple requests are represented by OPMs and complex requests by ISFs. If a request is fully specified, the Dialogue Manager sends the ISF or OPM to the Domain Knowledge Manager in order for a response to be retrieved from the domain agents. The Domain Knowledge Manager uses the request in the form of an ISF or OPM to make a recipe that describes how the requested information can be retrieved. The service-calls specified by the recipe are then executed and the result passed back to the Dialogue Manager in a specified response format, i.e. a feature structure containing specific fields like status, result, and error. The response is then used by the Dialogue Manager to communicate to the Generator what response it should present to the user.

In the description of the ÖTRAF system, the focus is on domain knowledge management but some aspects of dialogue management will also be described. Representations of requests are relevant since they serve as inputs to the Domain Knowledge Manager. An important part of the development of a dialogue system is to specify the types of requests and questions that should be handled, and the range of possible responses to these requests. Once this is done, customisation of the Dialogue Manager and the Domain Knowledge Manager can be done independently.

6.2 The Dialogue Manager

The requests the Dialogue Manager has to deal with can be divided into simple and complex requests. A number of complex information requests were discovered in the corpus, see chapter 4 section 4.2.1. Modelling of these is performed using the system task models, ISFs, that were introduced with the MALIN framework. The most common, trip information, occurs when the user needs to know how and when on a particular day, most often the current day, he or she can travel

from one point to another by public transportation. An ISF for such requests models information on departure and arrival destinations and information on arrival and/or departure time, which is necessary to access the timetable dabase. The user can also provide information about the travel type, but this is optional. Figure 6.2 shows an empty Trip ISF.

$$\begin{bmatrix} Type: & Trip \\ Arr: & req. \\ Dep: & req. \\ TTime: & req. \\ TType: & opt. \end{bmatrix}$$

Figure 6.2: An empty Trip ISF with fields for required information about arrival, departure and time, and optional information about travel type.

Another common information need, route information, is when the user wants information on which buses or trains go from one point to another. This ISF is similar to the Trip ISF but time information is no longer required, see figure 6.3

$$\begin{bmatrix} Type: & Trip \\ Arr: & req. \\ Dep: & req. \\ TTime: & opt. \\ TType: & opt. \end{bmatrix}$$

Figure 6.3: An empty Route ISF with fields for required information about arrival and departure, and optional information about time and travel type.

The simple requests are represented by OPM structures that describe objects and their properties and relations. In the ÖTraf system there are three types of OPMs: FIND_ALL, GET_VALUE, and INFO. The first type, FIND_ALL, is used to represent requests for objects that have some specific properties or relations to other

objects. An example of such requests is "Which buses pass the University?" (figure 6.4).

$$\left[\begin{array}{ll} Obj: & \left[\begin{array}{l} \#1\left[\ BusLine: \ ?\ \right] \\ \#2\left[\ Location: \ \left[\ Landmark: \ University\ \right]\ \right] \end{array}\right] \\ Prop: & \left[\ PassesBy: \ \left[\begin{array}{l} BusLine: \ \#1 \\ Location: \ \#2 \end{array}\right]\ \right] \end{array}\right]$$

Figure 6.4: The OPM representing the question "Which buses pass the University?"

Requests of the type GET_VALUE model questions where the value of a certain property or relation is sought. Figure 6.5 shows the example "How far is it from the City Centre to the Railway Station?".

$$\left[\begin{array}{ll} Obj: & \left[\begin{array}{l} \#1\left[\ Location: \ \left[\ Area: \ City\ Centre\ \right]\ \right] \\ \#2\left[\ Location: \ \left[\ Landmark: \ Railway\ Station\ \right]\ \right] \end{array}\right] \\ Prop: & \left[\ Distance: \ \left[\begin{array}{l} Location: \ \#1 \\ Location: \ \#2 \\ Value: \end{array}\right]\ \right] \end{array}\right]$$

Figure 6.5: The OPM representing the question "How far is it from the City Centre to the Railway Station?".

Finally the INFO type request is used when information about aspects not handled by the system, like prices or lost and found items, is requested. An example of request for price information is given in figure 6.6.

$$\left[\ Prop: \ \left[\ Aspect: \ Price\ \right]\ \right]$$

Figure 6.6: The OPM representing the question "How much is it?".

Both OPMs and ISFs are thus used to represent information requests, and are placed in the dialogue tree in order to keep track of what has been said. The use of the OPMs and ISFs will be further illustrated by the example dialogue in section 6.4.

6.3 The Domain Knowledge Manager

As mentioned in the description of the MALIN framework, the role of the Domain Knowledge Manager is to retrieve and integrate information from several domain agents in response to requests from the Dialogue Manager. Thus the Dialogue Manager forwards a fully specified request from the user in the form of an OPM or an ISF. These have a counterpart in the recipes used by the Domain Knowledge Manager. The recipes are thus application-specific and correspond to the information structures used by the Dialogue Manager.

6.3.1 Recipes

In the ÖTRAF system there are recipes corresponding to the two types of ISFs and various variants of OPMs.

Agent	*Service*	*Parameters*	*Result*
Spatial Reasoning Agent	findBusStops	Dep.BusStop, Dep.Landmark, Dep.Street, Dep.Area, Dep.Town	DepBusStops
Spatial Reasoning Agent	findBusStops	Arr.BusStop, Arr.Landmark, Arr.Street, Arr.Area, Arr.Town	ArrBusStops
Timetable Agent	getBusRoutes	DepBusStops, ArrBusStops, Date, ArrTime	Routes

Figure 6.7: An example of an uninstantiated recipe for route information.

The recipes for the Route and Trip ISFs are very similar, as can be seen in figure 6.7 and figure 6.8. Both need two sets of bus-stops corresponding to the departure and an arrival location, but only the Trip specification requires a time.

Agent	*Service*	*Parameters*	*Result*
Spatial Reasoning Agent	findBusStops	Dep.BusStop, Dep.Landmark, Dep.Street, Dep.Area, Dep.Town	DepBusStops
Spatial Reasoning Agent	findBusStops	Arr.BusStop, Arr.Landmark, Arr.Street, Arr.Area, Arr.Town	ArrBusStops
Temporal Reasoning Agent	findTime	TTime.Time	TravelTime
Timetable Agent	getTrips	DepBusStops, ArrBusStops, TravelTime	Trips

Figure 6.8: An example of an uninstantiated recipe for trip information.

Simple requests represented by OPMs are mapped to different recipes depending on the type of the OPM and the type of object or property that is requested. The recipes corresponding to the OPMs presented above illustrates this, see figure 6.9. The recipes for OPM requests are in general very simple and only consist of one service-call.

Agent	*Service*	*Parameters*	*Result*
Spatial Reasoning Agent	PassesBy	Objects.Location	BusLines

Agent	*Service*	*Parameters*	*Result*
Spatial Reasoning Agent	Distance	Objects.Location1, Objects.Location2	Distance

Agent	*Service*	*Parameters*	*Result*
System Information Agent	getInfo	Properties.Aspect	Info

Figure 6.9: Recipes corresponding to the OPMs requesting bus lines that pass a specific bus-stop, the distance between two locations, and price information.

6.3.2 Integration rules

The recipes describe how a request should be treated. But in cases where a request contain alternatives or ambiguities, as discussed in the chapter about MALIN, the request is divided into several recipes and the results from these have to be integrated. Different integration patterns are represented by integration rules. There are some very trivial domain- and application-independent rules, as described in section 5.3.2, but in the ÖTRAF system a specific type of ambiguity requires domain-dependent rules to be handled. This ambiguity arises when the same name can be used to denote objects of different types, as is examplified by the request "I want to go by bus to Berga Centre". It is ambiguous since "Berga Centre" is the name of both a bus-stop and a place. Does the user mean that she wants to go to the bus-stop called "Berga Centre" or that she wants to go to the place "Berga Centre"? If the system only considered the bus-

stop "Berga Centre", it would not find faster trips stopping at the bus-stop "Berga Söderleden" nearby "Berga Centre". Thus both possibilities are considered, but the interpretation of the location as the bus-stop is taken as the preferred interpretation. If both interpretations result in sets of trips these have to be compared and integrated. The integration rule used for this purpose is presented in figure 6.10. The getSet(<, travelTime, result2, result1) collects the set of trips in result2 (originating from the interpretation as a place) that has a travel time that is shorter than some of the trips in result1 (the bus-stop interpretation). These trips are returned as alternatives to the trips in result1.

Connector	*Conditions*	*Result*
POR	successful(result1.Status) successful(result2.Status) Exist getSet(<, travelTime, result2, result1)	Result = result1 Result.alternatives = getSet(<, travelTime, result2, result1)

Figure 6.10: An domain-specific integration rule used to integrate results from a trip request.

6.3.3 Domain agents

In the domain of local public transportation, four different domain agents are present. The **Temporal Reasoning Agent** contains a calendar and reasons about temporal expressions. The **Spatial Reasoning Agent** consists of a Geographical Information System and a reasoning mechanism used to deduce the relations between geographical objects. The **Timetable Agent** accesses an information source on the Internet which contains the timetables for local public transportation. There is also a **System and Help Information Agent** with system information, like references to human operators for questions outside the scope of timetable information, for example on lost property.

Temporal reasoning in dialogue systems is needed in several applications, and thus a number of approaches have been developed. Some of these will be described in relation to the Temporal Reasoning Agent. Spataial reasoning has not been explored to the same extent as temporal reasoning. A new approach [28], using a Geographical Information System, is therefore presented. The remaining two agents, are only briefly described.

6.3.4 Spatial Reasoning Agent

From the corpus presented in chapter 4 a number of relevant geographical and spatial objects, properties and relations have been identified. Thus knowledge about these types of objects and their properties and relations have been incorporated in the ÖTraf system in the form of a Spatial Reasoning Agent.

Spatial information

One common way of describing a departure or arrival location is by a town name, for example: "I wonder when the next bus from Skärblacka to Norrköping leaves?". This implies that the Spatial Reasoning Agent must be able to map a town name to a small set of bus-stops that can be used when searching the timetable database. Since small towns, in this case Skärblacka, often have few bus-stops and where only a few buses pass by, they can be mapped to a set containing all of the bus-stops in the town. Larger towns, Norrköping in the example, are more problematic to map to a set that contains few bus-stops. One solution is to ask the user for information that can be used to limit the possibilities. Another approach, which is similar to the approach used by the human operators, is to use the bus-stops that are most frequented by buses that travel between towns, such as the Bus Terminus or the Railway Station.

Another frequently used way to describe a location is to present an area such as a suburb. The problem is similar to the cases in which town names are used, sometimes further specification is needed in order to present a correct response, as in:

U2:	Hi, I wonder when the next bus from Malmslätt to the city runs?	Hej ja undrar nästa buss från Malmslätt in till stan går
S3:	Let's see, bus 213. Where in Malmslätt do you want to enter?	Då ska vi se här då buss tvåhundratretton går var i Malmslä vill du gå på

Some users present a location by providing the exact address, e.g. "Hi, I am going to 8 Owl street" ("Hej, jag vill åka till Ugglegatan 8"). For such examples the Spatial Reasoning Agent needs to map the address onto possible bus-stops near the address. A similar example is when a user presents two landmarks and expect the system to respond with a suggestion , e.g. "Hi, I would like to know how to get from the hospital down to IKEA in Linköping?" ("Hej, hur kan man ta sig från sjukhuset ner till IKEA i Linköping?").

The geographical objects and their properties are listed in figure 6.11. Between these objects various spatial relations exist, for example the distance between two objects or nearness of objects. The complete set of relations modelled in the ÖTRAF system is given in figure 6.12.

Reasoning about the relations between bus-stops and other geographical objects requires a representation of spatial and geographical information. Such information can be represented and reasoned with in different ways.

Representing geographical and spatial information

In the traditional quantitative approach a co-ordinate system is used as a basis for the representation of objects and regions. Information

Object	Properties
Bus-stop	Name, Id number, Location
Landmark	Name, Location
Street	Name, Location, Length
Area	Name, Location, Area
Town	Name, Location
Bus line	Line number

Figure 6.11: The set of geographical objects modelled in the ÖTraf system, and their properties.

Relation	Objects	Objects
Distance	Bus-stop, Landmark, Street, Area, Town	Bus-stop, Landmark, Street, Area, Town
Near	Bus-stop, Landmark, Street	Bus-stop, Landmark, Street
In	Bus-stop, Landmark, Street, Area	Area, Town
PassesBy	Bus line	Bus-stop
PassedBy	Bus-stop	Bus line

Figure 6.12: The spatial relations between the objects represented in the ÖTraf system.

about the objects' properties and relations among objects is extracted by means of arithmetic and trigonometrical computations. The representation and manipulation of spatial data is done numerically [45]. Another approach is to use a qualitative representation and reasoning mechanism. Qualitative representations are symbolic and based on discrete values: the distance between two objects can, for example, be expressed as one of the values very close, close, far or very far [16]. Maps are also a medium for representing geographic information. In a map both geometric aspects and symbolic representations of objects can be integrated [49].

In the field of Geographical Information Systems, quantitative representations are combined with maps. A geographic information system (GIS) can be defined as "A computer-based information system that enables capture, modelling, manipulation, retrieval, analysis, and presentation of geographically referenced data." [70, p. 1]. A GIS consists of three components: a database, an analytic engine and an interface. The design of the database affects how the GIS stores and models reality. The analytic engine is responsible for the manipulations and transformations of the data. The interface differs depending on the domain and application but one common feature is a map that can be used for visualisation of the spatial data in the database [68].

Using a GIS in a dialogue system that has to perform spatial reasoning has a number of advantages. For instance, a GIS comes with a variety of predefined functions for spatial reasoning. Furthermore, much spatial data is available in a form ready to use in a GIS. Since GISs support maps there is also the possibility of constructing a multimodal web-based interface, without modifying the underlying spatial representation.

The Spatial Reasoning Agent thus utilizes a GIS to represent the spatial information about bus-stops, streets, landmarks, areas and towns. Since traditional GISs are of a quantitative nature while spatial relations expressed in natural language are more qualitative, the Spatial Reasoning Agent must be able to transform qualitative concepts to quantitative. Two major concepts, *in* and *near*, which need to be transformed were identified in the corpus. The Spatial Reasoning Agent maps the qualitative term *near* onto a precise distance in the quantitative representation, i.e. into an area near the location. The concept *in* describes the topological relation between bus-stops, places or streets and a town or a suburb.

Services

Based on the geographical information held by the GIS and some reasoning mechanisms to derive new information from that represented the Spatial Reasoning Agent provides the following services: findBusStops, getBusStopsNear, BusStopNear?, getDistance, getPassesBy, PassesBy?, getPassedBy? and PassedBy?.

The corpus showed that a user's natural way of expressing a departure or arrival location is not by means of the "official" name of a bus-stop. Instead, other expressions are utilized, such as an area or town district, a set of reference points, or a street. Thus, the ÖTraf system must be able to map such an imprecise description to a set of bus-stops that corresponds to the description, i.e. the service findBusStops.

When the Spatial Reasoning Agent maps a departure or arrival location specified by the user in terms of a bus-stop, a street, a landmark, a suburb and/or a town to a set of bus-stops, it utilizes the *in* and *near* relations. All bus-stops within the distance which specify the *near* concept are gathered when mapping a landmark, street or bus-stop to the nearby bus-stops. For areas such as suburbs all the bus-stops that lie in the region are collected. The same applies for small towns. Large towns are mapped to the set of bus-stops which lie in the town and are key bus-stops, i.e. bus-stops that most of the bus routes pass. If a user has specified a location by means of different kinds of spatial information, for example a landmark and a street, the Spatial Reasoning Agent maps each kind of information separately to sets containing the nearby bus-stops and then takes the intersection of the sets. The complete algorithm used to find bus-stops is presented in figure 6.13.

```
Input: SpatialDescription, consisting of a Stop, Landmark,
Street, Area, and Town
Output: Bus-stops or ClarificationRequest

TempBusStops := null

for each SpatialUnit in SpatialDescription do

   if (SpatialUnit.isAmbiguous()) then
      SpatialUnit := disambiguate(SpatialUnit, SpatialDescription)

      if (SpatialUnit.isAmbiguous()) then
         Return ClarificationRequest(AMBIGUITY, SpatialUnit.getError())

   BusStopSet := findBusStops(SpatialUnit)

   if (BustStopSet.getError() != null) then
      Return ClarificationRequest(PROBLEM, BusStopSet.getError())
   else
      TempBusStops.add(BusStopSet)

BusStops := intersectionOf(TempBusStops)

if (NewBusStops == null) then
   Return ClarificationRequest(INCONSISTENCY,
                               findInc(SpatialDescription))

else if (BusStops.count() > limit) then
   Return ClarificationRequest(PROBLEM,
                               getProblem(SpatialDescription))

else
   Return NewBusStops
```

Figure 6.13: The algorithm used to map spatial descriptions to sets of bus-stops.

The *in* and *near* concepts are also used by the spatial reasoning module to resolve ambiguous spatial information which can refer to a number of different locations. When a name is ambiguous, the Spatial Reasoning Agent identifies the alternative locations and systematically examine how the alternatives are related to other spatial entities mentioned by the user.

In cases where a location is mapped to a too large set of bus-stops the Spatial Reasoning Agent must find a way of narrowing down the alternatives or ask the user for a specification. The mapping of streets to bus-stops that results in too many alternative bus-stops are treated in two different ways. If the street is long, i.e. more than 300 meters, the Spatial Reasoning Agent asks for a clarification by means of a bus-stop, landmark or suburb. Otherwise the Spatial Reasoning Agent presents the alternatives and asks the user to select one. Suburbs and areas are treated in a similar way. In the case of a large area, a bus-stop, landmark or street is requested for specification of the location. If the area is small, the user has to choose one bus-stop from the set of possible bus-stops.

The Spatial Reasoning Agent must also be able to discover inconsistent information that may be due to false presuppositions or misinterpretations of utterances. An example is a location specified in terms of a landmark and a town where there is no landmark with that name in the town. Using the *in* and *near* relations the Spatial Reasoning Agent can find the inconsistency and explain it to the user.

The getBusStopsNear service are very similar to findBusStops; it takes a bus-stop, landmark and/or street as input and use the same type of reasoning mechanisms to find the nearby bus-stops. The service can thus give as a result a set of bus-stops, a clarification request or an error message, depending on the provided description of the location. BusStopNear? is a variant of this service, but takes also a specific bus-stop as an argument and determines if this bus-stop is among the ones near the describe location.

The getDistance service take two locations as input and is expected to provide a distance. This service must also deal with possibly ambiguous or inconsistent descriptions.

The getPassedBy, getPassesBy, PassedBy? and PassesBy? simply retrieve the requested information from the GIS. The only problematic situation that occur is when a provided bus-stop has an ambiguous name. In these cases a clarification is sought from the user.

To summarise, the Spatial Reasoning Agent provides capabilities A16-A18, as it maps descriptions of locations to the precise entities, deals with ambiguous descriptions, and deals with inconsistent descriptions. It can also detect and deal with false presuppositions (A21).

6.3.5 Temporal Reasoning Agent

A study of the corpus from chapter 4 revealed several types of temporal expressions that need to be handled by the ÖTraf system.

Temporal expressions

Timepoints can be precise, e.g. "8.15" and "half past ten" ("halv elva"), but they can also be vague, for example "around four" (vid fyratiden), "by half past one approximately" ("vid halv två ungefär"). The precise timepoint can be used as is for database access; the vague timepoint needs, however, to be transformed into a precise time interval.

Common expressions in the corpus are "in the morning" (på morgonen), "afternoon" (eftermiddag), and "in the evening" ("kvällstid"). These are all examples of parts of day. How such expressions should be handled depends on whether other kinds of temporal information are available. If the user, for example, has said that she wants to

arrive at 8.15 in the morning, the information of part of day is used to disambiguate 8.15, concluding that it must be 8.15 a.m. On the other hand, if the user has given only "in the evening" as a temporal constraint for the trip, the part of day expression has to be mapped onto a precise time interval.

Information about the day or days, termed part of weeks, are sometimes given by the user in combination with other types of temporal descriptions. An example of such expressions found in the corpus are "weekdays Monday to Thursday" ("vardagkvällar måndag till torsdag"). This information is useful when route information is requested since buses may travel by different routes on different parts of the week, for example on the weekend.

Most of the requests found in the corpus concern trips taking place the same day, thus the date is left out. There are, however, some dialogues in which "tomorrow" ("imorgon") is used and there is also a case where a day is used to denote a date, e.g. "on Sunday" (på söndag). These types of expression have to be translated into exact dates in the system before accessing the database.

Common ways of modifying a temporal description is to use "before" ("före", "innan"), and "after" ("efter"). These modifiers transform a timepoint into a time interval with the timepoint as a starting- or ending point. Another type of modifier is "next" ("nästa") which also transform a timepoint, i.e. the current time, to an interval starting at the current timepoint.

Representing temporal information in dialogue systems

Many dialogue systems that provide timetable information, do scheduling, or negotiate times for meetings, have to deal with temporal expressions. Different approaches have been taken by different researchers but they all have much in common. In this section three approaches are described, one for bus timetables, one for train timetables, and one for negotiation of times for meetings.

Dobrin & Boda [23] describe how temporal expressions can be resolved in a www-based dialogue system for bus timetable inquiries. They propose a rule-based resolution for date and time expression. Three modules are used in the process: concept activation, labelling, and understanding.

The first stage, concept activation, involves a search of a database for words that match the input. The result is a subset of the words present in both the input and the database, changed to their canonical form.

The set of words that are passed on from the first stage are labelled in the next stage. The labels denote if a word is, for example, a number, a post-determiner, a predeterminer, a date determiner, a month determiner, or a week determiner. The expression "before 7.15" is labelled by HIGH_DET_BEFORE NUM PRE_DET_PAST NUM.

In the final stage rules are applied to the labelled time expressions. Lexical and contextual rules are associated with the different labels, and depending on the context, consisting of the two or three surrounding words, one rule is selected. The application of a rule mostly results in an assignment of a value to one or more fields in the formal time representation. For higher-order concepts one more step is needed. These concepts, e.g. next, before, after, are restrictions or constraints on previously processed and assigned values in the formal time representation. The constraints are often realised as a change from a specific timepoint to a time interval. For example, "before 7.15" is first mapped to [hh = 7 minimum = 15 durm = 0] and then modified to [hh = 5 minimum = 15 durm = 120], which is the full interpretation of the expression "before 7.15".

Hildebrandt et al. [37] also propose a three-step analysis of temporal expressions. The first step is to do a syntactic and semantic analysis at the phrase structure level. The second step involves interpretation at the sentence structure level. Finally the analysis sometimes has to be done at the dialogue level. Since they work within the domain of train timetable information, the resolution of a time expression results

in a form with four-by-two fields for day, month, hour, minute, and to and from respectively, see figure 6.14.

	From	Until
day		
month		
hour		
minute		

Figure 6.14: A form used to represent temporal information in a train timetable information system.

The syntactic analysis of time constituents in an utterance is done independently for each constituent. This analysis is followed by a semantic analysis of every time constituent. Primarily the core time of a time constituent is sought. That is, modifiers like "before" and "after" are overlooked. Then the time of reference is computed based on the modifiers and time of speech. The resulting time is an interval.

The analysis and interpretation on the sentence structure level mean that the separate time constituents have to be tested for consistency and merged into one. Ambiguity can often be resolved at this level, the time of day can, for example, be disambiguated using section of day.

When the system ask the user for confirmation of the computed time, the user can change or add some information. This has to be analysed on the phrase and sentence level, and then merged with the previous representation. This is performed on the dialogue level.

In the VERBMOBIL system temporal expressions are interpreted in a contextual evaluation module (ConEval) [62]. The interpretation is performed in two steps: the first translates the natural language input into a "ZeitGram" expression, and the next transforms this representation into a "interval description" (ID).

ZeitGrams are collections of temporal objects that have the form TYPE:VALUE. Possible temporal objects are points, intervals, modifiers, composed_pointlike, modifiable_pointlike, and modified modifiable_pointlike. The temporal objects can be qualified with the modifiers early, late, begin, middle, or end. They can also be "fuzzified" by a certain type of modifier, fuzzy.

Clock times (TOD) are represented like points of the form hh:mm. Timepoints can also be specified as a distance from a reference point. Intervals can be durations represented by a number and a temporal unit, limited time spans that are expressions of the form: before/after timepoint. Point_like objects are, for example, part of day (POD), day of week (DOW), part of year (seasons), and week of year.

The ZeitGrams are mapped to IDs that are typed feature structures consisting of two date expressions denoting the beginning and end of an interval. The date expressions have four attributes, year, month, day and time, that are further divided into subtypes. Thus this representation contains the same information as the representation presented by Hildebrandt et al. [37] although it is structured a bit differently. The IDs are constructed in a process where each temporal object in the ZeitGram is translated to an underspecified ID that is then unified with the ID built up so far. The translation of a temporal object to an ID involves knowledge from four sources. A table contains information for a heuristic mapping of part of day and part of year expressions to concrete intervals. A calendar provides information, such as date, about specific holidays or weeks. Utterances are time-stamped, and this information is used to deal with expression like today. Finally the context is used to resolve anaphoric expressions.

The three approaches are similar in several ways. The constituents of temporal expressions are first processed independently of each other. The final result is then composed by integrating the partial results. Modifiers that "fuzzify" a time are also added in the final stage in which they can transform a timepoint to a time interval. The format and information in the resulting temporal representation differ, however, between the approaches: the form presented in figure 6.14 is

very application-specific while the results from the other approaches are more general.

Services

Time expressions in the bus timetables are exact and precise. Time expressions in natural language are vague and imprecise. This discrepancy means that some sort of mapping between the two has to take place, the temporal expressions described above should be transformed to a date and a timepoint or a time interval.

In the ÖTraf system different features of the approaches presented in the previous section are used. The temporal descriptions sent by the Dialogue Manager to the Domain Knowledge Manager are represented in the same manner as ZeitGrams. The features partially overlap with the ones used in ZeitGrams; they are date, part of day (POD), part of week (POW), timepoint, time interval, and departure/arrival (DA). Timepoints can be made fuzzy by modifiers like around, before, after, earlier than and later than.

The representation of a resulting exact time, corresponding to a temporal description, is very similar to the one used by Dobrin & Boda [23]; it consists of a date and a time interval specified by a start timepoint and a duration.

Mapping between temporal descriptions and time representations is provided by the service findTime. As a basis for the service two knowledge sources are used. A database containing information on how part of day expressions can be mapped on to concrete intervals, and rules for how modifiers transform a timepoint to an interval are used. A calendar provides information that can be used to map part of week expressions on to precise dates. Following the approaches presented above, each constituent of a temporal expression is processed independently and result in a partial time representation. The correspondence between the different types of temporal expressions and time representations is shown in figure 6.15.

Type	Temporal expression	Time representation
Date	date(Month, Day)	DATE: YYMMDD
POD	pod(POD_STRING, MOD_STRING)	TIMEPOINT: HHMM DUR: MM
POW	pow(POW_STRING)	DATE: YYMMDD
Timepoint	timepoint(MOD_STRING, Hours, Minutes)	TIMEPOINT: HHMM DUR: MM
Time Interval	timeInterval(tp(...), tp(...))	TIMEPOINT: HHMM DUR: MM
DA	da(DA_STRING)	RIKT: DEP or ARR
POD_STRING:	morning, before_noon, noon, afternoon, evening, night, midnight	
POW_STRING:	Monday, Tuesday, ..., Sunday, weekdays, weekends, tomorrow, tomorrow+	
MOD_STRING:	around, now, earlier_than, later_than, before, after	
DA_STRING:	departure, arrival	

Figure 6.15: The figure shows the correspondence between vague temporal descriptions and the precise time representations used in the ÖTraf system.

In this first stage all timepoints are in the interval 8:00 to 19:59 if not explicitly indicated otherwise in the input. In the next stage, when the partial time representations are merged, the timepoint can be changed. If ambiguities or inconsistencies are detected after the merging of the partial results, the situation is analysed and the problem and a suggested solution, often in the form of a clarification, are given as the answer. The algorithm used to map temporal descriptions to time representations is presented in figure 6.16.

```
Input: TemporalDescription, consisting of a Date, POD, POW,
Timepoint, Time interval, and DA
Output: TimeRepresentation or ClarificationRequest

PartialTimeReps := null

for eah TemporalUnit in  TemporalDescription do
   PartialTimeRep := mapToPrecise(TemporalUnit)
   PartialTimeReps.add(PartialTimeRep)

TimeRepresentation = PartialTimeReps.getFirst()

for each PartialTimeRep in PartialTimeReps do
   TimeRepresentation := combine(TimeRepresentation,
                                 PartialTimeRep)

   if (TimeRepresentation == null) then
      Return ClarificationRequest(PROBLEM,
                                  TimeRepresentation.getError())

if (TimeRepresentation.isAmbiguous()) then
  Return ClarificationRequest(AMBIGUITY,
                              TimeRepresentation.getError())

if (TimeRepresentations.getDate() == null) then
  TimeRepresentation.addDefaultDate()

Return TimeRepresentation
```

Figure 6.16: The algorithm used to map temporal descriptions to time representations.

Thus the Temporal Reasoning Agent maps vague descriptions of temporal expressions to precise temporal entities (A16), and it also deals with ambiguous or erroneous descriptions (A17 and A18). It can also detect and handle false presuppositions (A21) concerning dates, for example if the user has said the 31st April, the system can inform her that April only has 30 days. Finally default information (A22) about dates can be added if not specified by the user.

6.3.6 Timetable Information Agent

When users have requested trip or route information, and the Spatial Reasoning Agent and the Temporal Reasoning Agent have provided precise departure and arrival locations and a time representation, the Timetable Information Agent is called. It accesses an information source available on the Internet and retrieves a set of trips or bus routes that match the parameters specified by the user. The retrieved information can be problematic in two ways that have to be dealt with by the Timetable Information Agent; there may be no answer that fulfil the specification (A5) or there may be too many trips or routes that correspond to the request (A6). If there is no answer the Domain Knowledge Manager can decide to relax some of the parameters and ask the agent to try again and find a trip or route. If there are too many answers, they are sorted according to a criterion, the default is the travel time, and only the best are returned.

6.3.7 System and Help Information Agent

The System and Help Information Agent use a database with pointers to other information sources, i.e. telephone numbers to various telephone-based services, to give helpful answers to questions that are slightly out of the range of the system, for example price information or lost and found items. The database also contains information about how to use the system and explanations of concepts used by the system. This agent thus provides capability A24 as it can communicate what the system can and cannot do.

6.4 An example dialogue

To illustrate how the Dialogue Manager and the Domain Knowledge Manager co-operate[1] to achieve a natural interaction with the user, consider the hypothetical dialogue shown in figure 6.17[2]. The dialogue tree in figure 6.18 shows the resulting structure of the dialogue.

U1: I want to go to the City Centre in Linköping.
S2: There are many bus-stops in the City Centre. Can you point on the map or give more specific information like a landmark or a street?
U3: Are there any bus-stops near the Garden Square?
S4: There are several bus-stops near the Garden Square.
< Shows the bus − stops in the map >
U5: Then I want to go there from the University.
S6: When do you want to go?
U7: On the 31st April before lunch.
S8: The 31st is not a valid date, there are only 30 days in April. Give a new date please.
U9: The 30th April.
S10: The trips are shown in the table.
< Shows a table of trips >

Figure 6.17: A hypothetical dialogue with the ÖTraf dialogue system.

The first utterance "I want to go to the City Centre", U1, is recognised as a request for information about trips and thus a Trip ISF is initiated. Information about the arrival location provided by the user is inserted in the ISF in the field Arr, which results in the structure presented in figure 6.19, which is included in IR1 in the dialogue tree.

[1] The example presented in this section is an extended version of the example in [29].

[2] The dialogue is constructed on the basis of the corpus in order to illustrate many of the features of the dialogue and domain knowledge management in our multi-modal system.

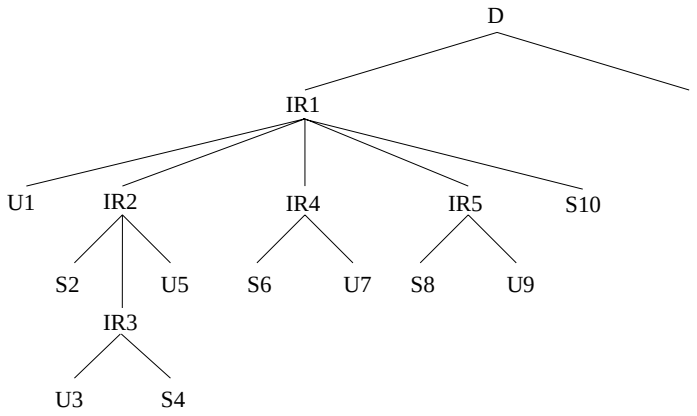

Figure 6.18: The dialogue tree resulting from the dialogue in figure 6.17.

$$\left[\begin{array}{ll} Type: & Trip \\ Arr: & [\ Area: \quad City\ Centre\ ] \\ Dep: & req. \\ TTime: & req. \\ TType: & opt. \end{array}\right]$$

Figure 6.19: The ISF in IR1 after processing U1.

The ISF indicates that information about the departure location and time has to be further specified by the user by the marker *req* in the fields Dep and TTime (TravelTime). However, before continuing the dialogue and asking the user for the information that is missing in the ISF, the Dialogue Manager asks the Domain Knowledge Manager to validate the values provided. This validation is performed to detect vague or erroneous information that might have been given by the user.

The arrival location in a Trip ISF will be used to find suitable bus-stops that can be used to search the timetable database. The valida-

tion of the arrival location therefore means that the Spatial Reasoning Agent tries to map the location to a small set of bus-stops. The Domain Knowledge Manager instantiates a recipe template for this type of questions, see 6.20.

Agent	*Service*	*Parameters*	*Result*
Spatial Reasoning Agent	findBusStops	Area: City Centre	BusStops

Figure 6.20: The recipe used to validate the arrival location of the request in figure 6.19.

The recipe only contains one service-call which is executed. In this case the Domain Knowledge Manager discovers that *Area: City Centre* is too vague a description since it corresponds to too many stops, in our case more than 5 stops. The Dialogue Manager is informed of this and is also given the information that more specific information like a point, a landmark or a street is required, figure 6.21.

$$\left[\begin{array}{ll} Status: & Error \\ Item: & [\ Area: \ City\ center\ ] \\ Type: & \left[\begin{array}{ll} TooMany: & BusStops \\ & [\ Up: \ 5\ ] \end{array}\right] \\ Solution: & \left[\begin{array}{ll} SpecInfo: & \{Point, \\ & Landmark, \\ & Street\} \end{array}\right] \end{array}\right]$$

Figure 6.21: The response from the Domain Knowledge Manager to the domain validation of the arrival location.

Thus, the user will not be asked to provide the value of another parameter since it would be an implicit confirmation that the arrival place is correct; instead a new IR-unit, IR2 in the dialogue tree, is created and a clarification, S2 "There are many bus-stops in the City Centre. Can you point on the map or give more specific information like a landmark or a street?", is initiated based on the information

from the Domain Knowledge Manager that indicates the problematic item, the type of problem, and a possible solution to the problem.

Instead of answering the system's question the user takes the initiative by requesting new information "Are there any bus-stops near the Garden Square?", U3. This request results in a new IR-unit, IR3, to be inserted in the dialogue tree as a clarification of the system's clarification in IR2, as shown in figure 6.18. The utterance is a simple request and the Dialogue Manager utilises an OPM to model this, figure 6.22.

$$\left[\begin{array}{ll} Obj: & \left[\begin{array}{ll} \#1 & \left[Stop: \ ? \right] \\ \#2 & \left[Landmark: \ Garden\ Square \right] \end{array}\right] \\ Prop: & \left[Near: \left[\begin{array}{ll} Location1: & \#1 \\ Location2: & \#2 \end{array}\right] \right] \end{array}\right]$$

Figure 6.22: The OPM in IR3 after processing of U3.

To answer this request means reasoning about spatial relations between geographical objects. The request is therefore sent to the Domain Knowledge Manager which retrieves and instantiates a recipe template for this type of question, see 6.23.

Agent	*Service*	*Parameters*	*Result*
Spatial Reasoning Agent	getBusStopsNear	Landmark: Garden Square	BusStops

Figure 6.23: The recipe corresponding to the request represented by the OPM in figure 6.22.

The recipe only contains one service-call which is executed. The request is successfully processed and some nearby bus-stops are found and sent back to the Dialogue Manager utilising the structure in figure 6.24.

$$\left[\begin{array}{lll} \textit{Status}: & \textit{Success} & \\ \textit{Stops}: & \left[\begin{array}{ll} \#1 & \left[\begin{array}{ll} \textit{Name}: & \textit{Centrum} \\ & \textit{Snickareg. } 30 \\ \textit{Id}: & 1268 \end{array}\right] \\ \#2 & \left[\begin{array}{ll} \textit{Name}: & \textit{Linnegatan} \\ \textit{Id}: & 1220 \end{array}\right] \\ \#3 & \left[\begin{array}{ll} \textit{Name}: & \textit{Stora torget} \\ \textit{Id}: & 450 \end{array}\right] \end{array}\right] \end{array}\right]$$

Figure 6.24: The response from the Domain Knowledge Manager to the OPM in IR3.

The Dialogue Manager can then ask the generator to present them to the user "There are several bus-stops near the Garden Square.", S4. The user responds to this answer by confirming his departure location "Then I want to go there from the University.", U5, and thereby responds to the request S2 of IR2. He also provides an arrival location. This new information, see figure 6.25, is placed in IR2.

$$\left[\begin{array}{ll} \textit{Obj}: & \left[\begin{array}{ll} \#1 & \left[\begin{array}{ll} \textit{Landmark}: & \textit{Garden} \\ & \textit{Square} \end{array}\right] \\ \#2 & \left[\begin{array}{ll} \textit{Landmark}: & \textit{University} \end{array}\right] \end{array}\right] \\ \textit{Prop}: & \left[\begin{array}{ll} \textit{Arr}: & \#1 \\ \textit{Dep}: & \#2 \end{array}\right] \end{array}\right]$$

Figure 6.25: The OPM in IR2 after processing U5.

The Dialogue Manager resumes processing of the ISF in IR1 and updates it with the arrival and departure location based on the information in the OPM of IR2. Information about the arrival location is added to the previously provided information in the field Arr. The new information about the departure location is inserted in the field Dep, yielding the structure in figure 6.26.

$$\left[\begin{array}{ll} Type: & Trip \\ Arr: & \left[\begin{array}{ll} Area: & City\ center \\ Landmark: & Garden\ Square \end{array}\right] \\ Dep: & \left[\ Landmark: \quad University\ \right] \\ TTime: & req. \\ TType: & opt. \end{array}\right]$$

Figure 6.26: The ISF in IR1 after updates with information from the subtree in IR2.

Again the Dialogue Manager asks the Domain Knowledge Manager for domain validation of the partially specified ISF. This involves mapping the departure location and the arrival location to sets of bus stops. The Domain Knowledge Manager therefore asks the Spatial reasoning Agent to execute service-call findBusStops with the parameters *Landmark: Garden Square* and *Landmark: University.*

Both service-calls are successful and the ISF is approved by the Domain Knowledge Manager. The Dialogue Manager now needs a time to complete the ISF, and consequently a new IR-unit, IR4 in the dialogue tree, is created and, in utterance S6 "When do you want to go?", the user is asked for this. The answer "On the 31st April before lunch.", U7, is a valid response to S6 and is inserted in the tree at IR4, see figure 6.27.

$$\left[\begin{array}{ll} Obj: & \#1\left[\begin{array}{ll} Date: & \left[\begin{array}{ll} Day: & 31 \\ Month: & April \end{array}\right] \\ Time: & \left[\begin{array}{ll} POD: & lunch \\ Mod: & before \end{array}\right] \end{array}\right] \\ Prop: & \left[\ TTime: \quad \#1\ \right] \end{array}\right]$$

Figure 6.27: The OPM in IR4 after processing U7.

The new information from IR4 is then inserted as TTime in the ISF of IR1. This results in a fully specified Trip ISF, figure 6.28.

$$\left[\begin{array}{ll} Type: & Trip \\ Arr: & \left[\begin{array}{ll} Area: & City\ center \\ Landmark: & Garden\ Square \end{array}\right] \\ Dep: & \left[\begin{array}{ll} Landmark: & University \end{array}\right] \\ TTime: & \left[\begin{array}{ll} Date: & \left[\begin{array}{ll} Day: & 31 \\ Month: & April \end{array}\right] \\ Time: & \left[\begin{array}{ll} POD: & lunch \\ Mod: & before \end{array}\right] \end{array}\right] \\ TType: & opt. \end{array}\right]$$

Figure 6.28: The ISF of IR1 after updates with information from IR4.

The ISF is again sent to the Domain Knowledge Manager for validation. A service-call to the findTime service provided by the Temporal Reasoning Agent is executed to test that the temporal description can be mapped to a time representation. When the Temporal Reasoning Agent tries to map the temporal description in TTime to a format suitable for timetable database search, it discovers the erroneous date. The Domain Knowledge Manager then returns a response, figure 6.29, to the Dialogue Manager informing it of the error. The Dialogue Manager initiates a new clarification IR-unit, IR5, and a clarification "The 31st is not a valid date, there are only 30 days in April. Give a new date please." is formulated, S8.

$$\left[\begin{array}{ll} Status: & Error \\ Item: & \left[\begin{array}{ll} Date: & \left[\begin{array}{ll} Day: & 31 \\ Month: & April \end{array}\right] \end{array}\right] \\ Type: & \left[\begin{array}{ll} NotValid: & \left[\begin{array}{ll} Month: & April \\ Up: & 30 \end{array}\right] \end{array}\right] \\ Solution: & \left[\begin{array}{ll} SpecInfo: & \{Date\} \end{array}\right] \end{array}\right]$$

Figure 6.29: The response from the Domain Knowledge Manager to the domain validation of the time description, explaining the erroneous date.

The user responds to the system's clarification request and provides a new date "The 30th April.", U9. The response (figure 6.30) is placed in IR5.

$$\left[\begin{array}{ll} Obj: & \#1\left[\begin{array}{ll} Date: & \left[\begin{array}{ll} Day: & 30 \\ Month: & April \end{array}\right] \end{array}\right] \\ Prop: & \left[\begin{array}{ll} TTime: & \#1 \end{array}\right] \end{array}\right]$$

Figure 6.30: The OPM of IR5 after U9.

The information in the clarification request IR-unit, IR5, is propagated to the ISF of IR1 which is updated. This time the new information replaces the old in TTime since it was erroneous. The resulting ISF is presented in figure 6.31.

$$\left[\begin{array}{ll} Type: & Trip \\ Arr: & \left[\begin{array}{ll} Area: & City\ Centre \\ Landmark: & Garden\ Square \end{array}\right] \\ Dep: & \left[\begin{array}{ll} Landmark: & University \end{array}\right] \\ TTime: & \left[\begin{array}{ll} Date: & \left[\begin{array}{ll} Day: & 30 \\ Month: & April \end{array}\right] \end{array}\right] \\ Time: & \left[\begin{array}{ll} POD: & lunch \\ Mod: & before \end{array}\right] \\ TType: & opt. \end{array}\right]$$

Figure 6.31: The ISF of IR1 after integration with the information in IR5.

Once more a validation of the ISF is performed by the Domain Knowledge Manager. This time no problems are detected and a search for suitable trips can finally be done.

The Domain Knowledge Manager does this by instantiating a Trip recipe, figure 6.32, and then execute the service-calls one by one: first asking the Spatial Reasoning Agent to map the departure and arrival locations to two sets of bus-stops, then asking the Temporal Reasoning Agent to map the vague temporal description to a precise time

Agent	*Service*	*Parameters*	*Result*
Spatial Reasoning Agent	findBusStops	_, University, _, _, _	DepBusStops
Spatial Reasoning Agent	findBusStops	_, Garden Square, _, _, _	ArrBusStops
Temporal Reasoning Agent	findTime	date(30, April), pod(lunch, before), _, _, _	TravelTime
Timetable Agent	getTrips	DepBusStops, ArrBusStops, TravelTime	Trips

Figure 6.32: The instantiated recipe for trip information corresponding to the request represented by the ISF in figure 6.31.

interval. Given this information, the Domain Knowledge Manager then searches the timetable database to find one or more trips that fulfil the requirements. The resulting trips are sent back to the Dialogue Manager and displayed to the user, "The trips are shown in the table.", S10.

6.5 Summary

The main purpose of this chapter was to illustrate how a Domain Knowledge Manager can be implemented and integrated in a dialogue system. As illustrated by the dialogue in section 6.4 the Dialogue Manager and Domain Knowledge Manager divide the labour and provide different capabilities needed to achieve natural interaction with the user. To summarise, the Dialogue Manager utilises the system task models, the ISFs, to handle tasks and requests (capabilities A1-A4), the dialogue model to allow mixed-initiative dialogue (capabilities A9-A11), and the dialogue history to follow shifts in fo-

cus and resolve anaphora and ellipsis (capabilities A12-A13). The Domain Knowledge Manager utilises domain agents to handle problematic responses from background systems (capabilities A5-A6), to handle descriptions (capabilities A16-A18), domain knowledge reasoning (capabilities A19-A22). Thus a natural separation of knowledge sources and capabilities related to dialogue management and domain knowledge management has been achieved.

Chapter 7

Summary and Discussion

In this thesis a new concept called **domain knowledge management** has been introduced. Domain knowledge management includes issues related to representation and use of domain knowledge and access of background systems, issues that previously have been incorporated in dialogue management.

The work on domain knowledge management presented in this thesis can be divided in two parts. On a general theoretical level, knowledge sources and models used for dialogue management, including domain knowledge management, have been studied and related to the capabilities they support. On a more practical level, domain knowledge management has been discussed in the context of the MALIN dialogue system framework and a specific instance of this framework, the ÖTRAF system. In these, domain knowledge management has been implemented in a separate module, a Domain Knowledge Manager.

7.1 Knowledge sources and capabilities

To clarify what dialogue management involves and what can be separated out in domain knowledge management, the knowledge sources and models used for dialogue management in dialogue systems, and the capabilities they support were analysed.

In a survey of eight information-providing or problem-solving dialogue systems four types of knowledge, represented by seven different knowledge sources, were identified. **Dialogue knowledge** is represented in *dialogue models* and *dialogue histories*. **Task knowledge** can be divided in *system task models* and *user task models*. **Domain knowledge** is held by *domain knowledge sources* and *conceptual models*. Finally, **Knowledge of the user** can be represented by a *user model*. With this new categorisation the sometimes confusing terminology concerning knowledge sources and models in dialogue systems was made more clear.

A list of capabilities considered useful for achieving natural and graceful interaction was compiled from a set of guidelines and development principles for dialogue system (see Appendix A). The relation between the knowledge sources and the capabilities were mapped out and summarised in table in Appendix B. This information can be used to support design of dialogue systems as it forces the developer to make explicit choices of which capabilities to include in the system and consider the limitations the design of knowledge sources and models impose on the system.

7.2 Future work on knowledge sources and capabilities

The guidelines and development principles used to create the list of capabilities are primarily aimed at information-providing dialogue systems. To make the list complete and also include other types of

dialogue systems, more work has to be done. For this purpose more guidelines and development principles can be examined, and studies of corpora will also be useful.

As mentioned above, the capabilities can be used for the design of dialogue systems. Another way to use them is for evaluation of dialogue systems and frameworks. Evaluation can be made for various purposes: to detect errors in design and implementation during the development, to measure how well it meets the user's needs and expectations in the actual context of use, and to obtain quantitative measurements for comparisons with other systems [9]. How dialogue systems and dialogue managers can and should be evaluated is a difficult question to answer.

An approach to evaluation of dialogue systems proposed in the TRINDI project [65] is to examine whether certain characteristic behaviours are manifested by the system. A "Tick-list" consisting of twelve yes-no questions is presented and suggested for this purpose [17]. Some of these questions correspond to the capabilities, for example "Can the system deal with answers to questions that give more information than was requested?" corresponds to capability A9, "To allow the user to over-answer questions" and "Can the system deal with ambiguous designators?" corresponds to capability A17, "To detect ambiguous descriptions and deal with them". Another use of the capabilities, besides design of dialogue systems, could therefore be to create "tick-lists" that can be used for evaluation and comparison of dialogue systems and frameworks.

7.3 The Domain Knowledge Manager

If we examine table in appendix B we see that all of the capabilities related to mixed-initiative dialogue and handling of focus and discourse, and most of the capabilities for tasks and requests, rely only on knowledge sources and models for dialogue and task knowledge. Furthermore, almost all of the capabilities related to domain knowl-

edge depend exclusively on knowledge of the domain. This grouping of capabilities and knowledge sources was seen as a support for a separation between dialogue management and domain knowledge management.

In chapter 5 a separate module for domain knowledge management, a **Domain Knowledge Manager**, was introduced in the context of the MALIN dialogue system framework. The Domain Knowledge Manager co-operates with the Dialogue Manager and provides the capabilities needed to handle problematic responses to requests (A5-A6), to handle descriptions (A16-A18), to handle false presuppositions (A21), to perform domain reasoning (A20), to provide default information (A22), and to have system meta-knowledge(A24).

A few previous attempts to separate issues related to domain knowledge management from dialogue management have been made. For example, Whittaker & Attwater [69], suggested a new component, the Information Manager, that maps descriptions provided by the Dialogue Manager to a suitable representation for the external database. Another example is the Action Manager proposed by Radev et al. [52]. In their architecture a requested action is delivered to the Action Manager and it has to decide how the action should be executed and which source to access. It is, however, the Dialogue Manager that is responsible for the translation between the requests and the actions carried out by the Action Manager. Thus, both these approaches only deal with very few of the domain knowledge management issues, or a monolithic application. None of them handles all the issues and the integration of several domain knowledge sources. A more sophisticated and general module is needed to handle all these features.

This type of more sophisticated module can be found in the TRIPS dialogue shell [6], where a Behavioural Agent separated from dialogue management is responsible for communication with the back-end system. The separation of dialogue management and domain knowledge management is, however, not complete as the Behavioural Agent is able to ignore requests from the Discourse Manager and instead give information that it considers more important, thus, taking on some of the dialogue management responsibilities. In the MALIN framework

all dialogue management features are handled by the Dialogue Manager and the Domain Knowledge Manager only responds to requests made by the Dialogue Manager.

The use of a specialised Domain Knowledge Manager has a number of advantages. The first is that dialogue management becomes more focused as it only has to consider dialogue phenomena, while domain-specific reasoning is handled by the Domain Knowledge Manager. The second major advantage is that once an interface between the Dialogue Manager and the Domain Knowledge Manager has been specified, they can be developed and experimented with independently of each other. This in turn facilitates porting of a system to new domains since domain-related issues are included in the domain knowledge sources. Another advantage is that the domain knowledge sources can be easily modified, exchanged, and reused. Finally, with a separate module for domain knowledge management the domain-dependent features and background systems are gathered in one location, which can be used both by the Dialogue Manager and also the Interpreter and Generator.

The design of the Domain Knowledge Manager can be designed has two important features: the mechanisms for managing requests are generic and to some extent domain-independent, and the domain knowledge sources have a common well-specified interface based on an agent communication protocol. This means that a system can be easily extended by introducing new domain knowledge sources and adding new recipe templates and integration rules; porting to a new domain for the Domain Knowledge Manager only involves the creation of new recipe templates, integration rules, and plugging in new domain agents.

7.4 Future work on the Domain Knowledge Manager

Future work on the Dialogue Manager can be made in three different directions: new capabilities and knowledge sources for information-providing dialogue systems, new types of domains for information-providing dialogue systems, and new service types such as tutoring or solving problems.

The proposed Domain Knowledge Manager and the dialogue system framework MALIN was developed with certain types of services and domains in mind. Depending on the capabilities supported by information-providing dialogue systems a possible classification could be: simple, intelligent, and adaptive.

Simple information-providing dialogue systems handle fairly simple information requests that do not require sophisticated task and domain knowledge. The LINLIN framework falls into this category.

Intelligent information-providing dialogue systems handle more complex information requests, which rely on sophisticated task and domain knowledge. The MALIN framework is an example of this class.

Adaptive information-providing dialogue systems do not only handle complex information requests but can also adapt to the user, thus requiring knowledge of the user as well as the tasks and the domain.

Based on this hierarchy a natural next step would be to extend the framework so as to also handle dialogues where the system can adapt to the user. The question is whether this adaptation is part of dialogue management or domain knowledge management. If the user model is utilised to guide the dialogue, for example which information is requested from the user by the system or how the user is prompted,

it should belong to dialogue management. However, if knowledge of the user's expertise is used to decide what information should be retrieved from background system or how descriptions and request responses should be handled, it should be part of domain knowledge management. Since both uses can and probably will occur, the user model should probably be a knowledge source shared by the Dialogue Manager and the Domain Knowledge Manager.

It would also be of interest to study how the Domain Knowledge Manager can be modified to work in new types of domains where the domain is more dynamic and less structured, for example large document bases and unstructured information. Applications of this type may include methods like data mining, information extraction, and automatic generation of ontologies. The role of the Domain Knowledge Manager would be to serve as the intermediator between the Dialogue Manager and an information processing component responsible for bringing structure to the documents and the information. Advantages of using a Domain Knowledge Manager are that the Dialogue Manager need not be aware of the unstructured nature of the domain and consider this when contextually specifing a request, and that an information processing component need not consider aspects of the dialogue.

Another way of extending the framework is to widen the type of domains and the type of services the framework supports. Moving from information-providing dialogue systems to other types of services like problem-solving and tutoring systems, domain knowledge reasoning will probably become even more important. In these types of systems it is crucial that the system can reason about and integrate knowledge from several domain knowledge sources. New capabilities, for example explanation, will be required, and with them it is likely that new types of knowledge sources will have to be added. An interesting question to examine is how the Domain Knowledge Manager should be modified to also handle these new types of services.

Bibliography

[1] Alicia Abella, Michael K. Brown, and Bruce Buntschuh. Development principles for dialog-based interfaces. In Elisabeth Maier, Marion Mast, and Susann Luperfoy, editors, *Dialogue Processing in Spoken Language Systems*, volume 1236 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, 1997.

[2] Lars Ahrenberg, Arne Jönsson, and Nils Dahlbäck. Discourse representation and discourse management for natural language interfaces. In *Proceedings of the Second Nordic Conference on Text Comprehension in Man and Machine, Täby, Sweden*, 1990.

[3] Jan Alexandersson. Plan recognition in VERBMOBIL. Technical Report 81, DFKI GmbH,
URL: http://www.dfki.uni-sb.de/cgi-bin/verbmobil/htbin/doc-access.cgi, 1995.

[4] Jan Alexandersson, Elisabeth Maier, and Norbert Reithinger. A robust and efficient three-layered dialogue component for speech-to-speech translation system. Technical Report 50, DFKI GmbH,
URL: http://www.dfki.uni-sb.de/cgi-bin/verbmobil/htbin/doc-access.cgi, 1994.

[5] Jan Alexandersson and Norbert Reithinger. Designing the dialogue component in a speech translation system. In *Proceedings*

of the Ninth Twente Workshop on Language Technology (TWLT-9), pages 35–43, 1995.

[6] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, , and Amanda Stent. An architecture for a generic dialogue shell. *Journal of Natural Language Engineering, Special Issue on Best Practices in Spoken Language Dialogue Systems Engineering*, 3(6):1–16, December 2000.

[7] James Allen, Lenhart Schubert, George Ferguson, Peter Heeman, Chung He Hwang, Tsuneaki Kato, Mark Light, Nathaniel Martin, Bradford Miller, Massimo Poesio, and David Traum. The TRAINS project: a case study in building a conversational planning agent. *Journal of Experimental and Theoretical Artificial Intelligence*, 7:7–48, 1995.

[8] S. Bennacef, L. Devillers, S. Rosset, and L. Lamel. Dialog in the RAILTEL telephone-based system. In *Proceedings of International Conference on Spoken Language Processing, ICSLP'96*, volume 1, pages 550–553, Philadelphia, USA, October 1996.

[9] Niels Ole Bernsen, Hans Dybkaer, and Laila Dybkaer. *Designing Interactive Speech Systems: From First Ideas to User Testing.* Springer Verlag, 1998.

[10] Eric Bilange. A task independent oral dialogue model. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics, EACL'91*, pages 83–88, Berlin, Germany, 1991.

[11] Rolf Carlson and Sheri Hunnicut. Generic and domain-specific aspects of the Waxholm NLP and dialog modules. In *Proceedings of International Conference on Spoken Language Processing, ICSLP'96*, volume 2, pages 677–680, Philadelphia, USA, October 1996.

[12] Rolf Carlson, Sheri Hunnicut, and Joakim Gustafsson. Dialog management in the Waxholm system. In *Papers from the Eighth Swedish Phonetics Conference, Working Papers 43*, pages 46–49, 1994.

[13] Philip. R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[14] Robin Cohen. On the relationship between user models and discourse models. *Computational Linguistics*, 14(3):88–90, 1988.

[15] Robin Cohen, Coralee Allaby, Christian Cumbaa, Mark Fitzgerald, Konsin Ho, Bowen Hui, Celine Latulipe, Fletcher Lu, Nancy Moussa, David Pooley, Alex Qian, and Saheem Siddiqi. What is initiative. *User Modeling and User-adapted Interaction*, 8(3–4):5–48, 1998.

[16] A.G. Cohn. Qualitative spatial representation and reasoning techniques. In G. Brewka, C. Habel, and B. Nebel, editors, *KI-97: Advances in Artificial Intelligence: Proceedings of 21st Annual German Conference on Artificial Intelligence*, volume 1303 of *Lecture Notes in Artificial Intelligence*, pages 1–30. Springer-Verlag, 19.

[17] Robin Cooper, Staffan Larsson, C. Matheson, and David Traum. Coding instructional dialogue for information state. Technical Report D1.1, TRINDI,
URL: http://www.ling.gu.se/research/projects/trindi/publications.html, 2000.

[18] The CSLU toolkit.
URL: http://cslu.cse.ogi.edu/toolkit/, 2001.

[19] Nils Dahlbäck and Arne Jönsson. Knowledge sources in spoken dialogue systems. In *Proceedings of Eurospeech'99*, Budapest, Hungary, 1999.

[20] Nils Dahlbäck. *Representations of Discourse, Cognitive and Computational Aspects*. PhD thesis, Linköping University, 1991.

[21] Nils Dahlbäck. Towards a dialogue taxonomy. In Elisabeth Maier, Marion Mast, and Susann LuperFoy, editors, *Dialogue Processing in Spoken Language Systems*, number 1236 in LNAI-Lecture Notes in Artificial Intelligence. Springer Verlag, 1997.

[22] Nils Dahlbäck and Arne Jönsson. Integrating domain specific focusing in dialogue models. In *Proceedings of Eurospeech'97*, volume 4, pages 2215–2218, Rhodes, Greece, 1997.

[23] Cristina Dobrin and Peter Boda. Resolution of date and time expressions in a www-based dialogue system. In *COST249 10th Management Committee Meeting*, Porto, Portugal, February 12-13 1998.

[24] George Ferguson, James Allen, and Brad Miller. TRAINS-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third Conference on Artificial Intelligence Planning Systems, AIPS-96*, pages 70–77, 1996.

[25] George M. Ferguson, James F. Allen, Brad W. Miller, and Eric K. Ringger. The design and implementation of the TRAINS-96 system: A prototype mixed-initiative planning assistant. TRAINS Technical Not 96-5, October 1996.

[26] Annika Flycht-Eriksson. A survey of knowledge sources in dialogue systems. In *Proceedings of IJCAI'99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 41–48, Stockholm, August 1999.

[27] Annika Flycht-Eriksson. A domain knowledge manager for dialogue systems. In *Proceedings of the 14th European Conference on Artificial Intelligence, ECAI 2000*. IOS Press, Amsterdam, 2000.

[28] Annika Flycht-Eriksson and Arne Jönsson. A spoken dialogue system utilizing spatial information. In *Proceedings of International Conference on Spoken Language Processing, ICSLP'98*, volume 4, pages 1207–1210, Sydney, Australia, December 1998.

[29] Annika Flycht-Eriksson and Arne Jönsson. Dialogue and domain knowledge management in dialogue systems. In *Proceedings of 1st SIGdial Workshop on Discourse and Dialogue*, Hong Kong, 2000.

[30] David Goddeau, Eric Brill, James Glass, Christine Pao, Michael Philips, Joseph Polifroni, Stephanie Seneff, and Victor Zue.

GALAXY: A human-language interface to on-line travel information. In *Proceedings of International Conference on Spoken Language Processing, ICSLP'94*, pages 707–710, Yokohama, Japan, September 1994.

[31] Paul H. Grice. Logic and conversation. In Peter Cole and Jerry L. Morgan, editors, *Syntax and Semantics (vol. 3) Speech Acts*. Academic Press, 1975.

[32] Barbara J. Grosz and Candace L. Sidner. Attention, intention and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[33] Eli Hagen. An approach to mixed initiative spoken information retrieval dialogue. *User modeling and User-Adapted Interaction*, 9(1-2):167–213, 1999.

[34] Frank Seide Harald Aust, Martin Oerder and Volker Steinbiss. The Philips automatic train timetable information system. *Speech Communication*, 1995.

[35] Philip J. Hayes and D. Raj Reddy. Steps toward graceful interaction in spoken and written man-machine communication. *International Journal of Man-Machine Studies*, 19:231–284, 1983.

[36] Paul Heisterkamp, Scott McGlashan, and Nick Youd. Dialogue semantics for a spoken dialogue system. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP'92*, Banff, Canada, 1992.

[37] Bernd Hildebrandt, Gernot A. Fink, Franz Kummert, and Gerhard Sagerer. Understanding of time constituents in spoken language dialogues. pages 939–942, 1994.

[38] Arne Jönsson. *Dialogue Management for Natural Language Interfaces*. PhD thesis, Linköping University, 1993.

[39] Arne Jönsson. A model for habitable and efficient dialogue management for natural language interaction. *Natural Language Engineering*, 3(2/3):103–122, 1997.

[40] Arne Jönsson and Nils Dahlbäck. Talking to a computer is not like talking to your best friend. In *Proceedings of the First Scandinavian Conference on Artificial Intelligence, Tromsø*, 1988.

[41] Arne Jönsson and Lena Strömbäck. Robust interaction through partial interpretation and dialogue management. In *Proceedings of Coling/ACL'98, Montréal*, 1998.

[42] Robert Kass and Tim Finin. Modeling the user in natural language systems. *Computational Linguistics*, 14(3):5–22, 1988.

[43] Alfred Kobsa. User models and discourse models united they stand... *Computational Linguistics*, 14(3):91–94, 1988.

[44] Staffan Larsson, Lena Santamarta, and Arne Jönsson. Using the process of distilling dialogues to understand dialogue systems. In *Proceedings of 6th International Conference on Spoken Language Processing,ICSLP2000*, Beijing, China, 2000.

[45] Robert Laurini and Derek Thompson. *Fundamentals of Spatial Information Systems.* Academic Press, 1992.

[46] David L. Martin, Adam Cheyer, and Gowang Lo Lee. Agent Development Tools for the Open Agent Architecture. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 387–404, London, April 1996. The Practical Application Company Ltd.

[47] Scott McGlashan, Norman Fraser, Nigel Gilbert, Eric Bilange, Paul Heisterkamp, and Nick Youd. Dialogue management for telephone information systems. In *Proceedings of the International Conference on Applied Language Processing, ICSLP'92*, Trento, Italy, 1992.

[48] Michael McTear. Spoken dialogue technology: Enabling the conversational user interface.
URL: http://www.infj.ulst.ac.uk/~ cbdg23/survey/spoken_dialogue_technology.html, 2000.

[49] M. Monmonier. *How to Lie with Maps.* Univ. of Chicago Press, second edition, 1996.

[50] Martha E. Pollack. Plans as complex mental attitudes. In *Intentions in Communication*, chapter 5. MIT Press, 1990.

[51] Joachim Quantz, Manfred Gehrke, Uwe Kssner, and Birte Schmitz. The VERBMOBIL domain model version 1.0. Technical Report 29, Technishe Universität Berlin, September 1994.

[52] Dragomir R. Radev, Nanda Kambhatla, Catherine Wolf Yiming Ye, and Wlodek Zadrozny. Dsml: A proposal for xml standards for messaging between components of a natural language dialogue system. In *Proceedings of AISB Workshop on Reference Architectures and Data Standards for NLP*, Edinburgh, UK, April 1999.

[53] Elaine Rich. User modelling via stereotypes. In *Readings in Intelligent User Interfaces*, pages 329–341. Morgan Kaufmann, 1998.

[54] A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. Creating natural dialogs in the Carnegie Mellon Communicator system. In *Proceedings of Eurospeech'99*, volume 4, pages 1531–1534, 1999.

[55] Stuart Russel and Peter Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, 1995.

[56] Emanuel A. Schegloff and Harvey Sacks. Opening up closings. *Semiotica*, 7:289–327, 1973.

[57] Ethel Schuster. Establishing the relationship between discourse models and user models. *Computational Linguistics*, 14(3):82–85, 1988.

[58] Stephanie Seneff, David Goddeau, Christine Pao, and Joseph Polifroni. Multimodal discourse modelling in a multi-user multi-domain environment. In *Proceedings of International Conference on Spoken Language Processing, ICSLP'96*, pages 192–195, Philadelphia, USA, October 1996.

[59] Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. GALAXYII: A reference architecture for conversational system development. In *Proceedings*

of International Conference on Spoken Language Processing, ICSLP'98, volume 3, pages 931–934, Sydney, Australia, December 1998.

[60] R. W. Smith and D. R. Hipp. *Spoken Natural Language Dialog Systems: A Practical Approach.* New York: Oxford University Press, 1994.

[61] Ronnie W. Smith. Integration of domain problem solving with natural language dialog: The missing axiom theory. In *Proceedings of Applications of AI X: Knowledge-Based Systems*, pages 270–278, 1992.

[62] Manfre Stede, Stefan Haas, and Uwe Küssner. Tracking and understanding temporal descriptions in dialogue. Technical Report 232, Technishe Universität Berlin, October 1998.

[63] Lena Strömbäck and Arne Jönsson. Robust interpretation for spoken dialogue systems. In *Proceedings of ICSLP'98, Sydney, Australia*, 1998.

[64] David R. Traum. Conversational agency: The TRAINS-93 dialogue manager. In Susann LuperFoy, Anton Nijholt, and Gert Veldhuijzen van Zanten, editors, *Proceedings of Twente Workshop on Language Technology, TWLT-II*, 1996.

[65] The TRINDI project website.
URL: http://www.ling.gu.se/research/projects/trindi/, 2001.

[66] The TRINDIKIT.
URL: http://www.ling.gu.se/research/projects/trindi/trindikit/, 2001.

[67] UMBC KQML Web.
URL: http://www.cs.umbc.edu/kqml/, 2000.

[68] W.P.A. van Deursen. *Geographical Information Systems and Dynamic Models.* PhD thesis, University Utrecht, 1995.

[69] Steve J. Whittaker and David J. Attwater. The design of complex telephony applications using large vocabulary speech technology. In *Proceedings of International Conference on Spoken*

Language Processing, ICSLP'96, pages 705–708, Philadelphia, USA, October 1996.

[70] Michael F. Worboys. *GIS: A Computing Perspective*. Taylor & Francis, 1997.

Appendix A

Capabilities for dialogue management

This appendix contains the collection of capabilities considered useful for achieving natural interaction between a user and a dialogue system.

Handling tasks and requests

A1 To identify the task.

A2 To identify sub-tasks and know how they are related to a task.

A3 To reason about how much of a task has been achieved so far.

A4 To decide what action to take in order to achieve a task.

A5 To deal with situations in which no answer can be retrieved from the background system.

A6 To deal with situations in which the answer from the background system includes to much information.

A7 To detect and deal with hypothetical questions.

A8 To explicitly communicate a commitment made by the user in the conversation.

Achieving mixed-initiative dialogue

A9 To allow the user to over-answer questions.

A10 To allow the user to initiate clarification sub-dialogues.

A11 To allow the user to abandon the current request and pose a new request instead.

Handling focus and discourse

A12 To follow shifts in focus.

A13 To resolve anaphora and ellipsis.

A14 To answer questions on what has been said and done during the conversation

A15 To answer questions about the reason why an action was performed.

Handling domain knowledge

A16 To map a description to an entity.

A17 To detect ambiguous descriptions and deal with them.

A18 To detect erroneous descriptions and deal with them.

A19 To know the type and structure of the entities in the domain.

A20 To reason about and derive new information from the information provided by the user.

A21 To deal with a user's erroneous inferences or false presuppositions

A22 To have domain-related default information.

A23 To adapt to the user's domain expertise.

A24 To know what the system can and cannot do.

Appendix B

Capabilities and knowledge sources

The relations between capabilities and knowledge sources and models are summarised in the tables on the following pages. Knowledge sources that are Required to achieve a specific capability are marked by an R and L means at Least one of the knowledge sources is required.

	Knowledge source/ Capability	Dial Mod	Dial Hist	Sys Task Mod	User Task Mod	Dom Mod	Conc Mod	User Mod
	Task and requests							
A1	To identify the task.	R		L	L			
A2	To identify sub-tasks and know how they are related to a task.	L		L	L			
A3	To reason about how much of a task has been achieved so far.		L	L	L			
A4	To decide what action to take in order to achieve a task.	R	L	L	L			
A5	To deal with situations in which no answer can be retrieved from the background system.	R		L		L	L	
A6	To deal with situations in which the answer from the background system includes to much information.	R		L		L	L	
A7	To detect and deal with hypothetical questions.	R	R					
A8	To explicitly communicate a commitment made by the user in the conversation.	R	L	L	L			
	Mixed-initiative dialogue							
A9	To allow the user to over-answer questions.	R	L	L	L			
A10	To allow the user to initiate clarification sub-dialogues.	R	R					
A11	To allow the user to abandon the current request and pose a new request instead.	R						
	Focus and discourse							
A12	To follow shifts in focus.	R	R					
A13	To resolve anaphora and ellipsis.		R					
A14	To answer questions on what has been said and done during the conversation	R	R	L	L			
A15	To answer questions about the reason why an action was performed.	R	L		L			

	Knowledge source/ Capability	Dial Mod	Dial Hist	Sys Task Mod	User Task Mod	Dom Mod	Conc Mod	User Mod
	Domain knowledge							
A16	To map a description to an entity.					R		
A17	To detect ambiguous descriptions and deal with them.					L	L	
A18	To detect erroneous descriptions and deal with them.					L	L	
A19	To know the type and structure of the entities in the domain.					L	L	
A20	To reason about and derive new information from the information provided by the user.					L	L	
A21	To deal with a user's erroneous inferences or false presuppositions	R	L		L	L		
A22	To have domain-related default information.			L		L		
A23	To adapt to the user's domain expertise.							R
A24	To know what the system can and cannot do.					L	L	

LINKÖPINGS UNIVERSITET

Avdelning, institution
Division, department

Institutionen för datavetenskap

Department of Computer and Information Science

Datum
Date

2001-05-07

Språk
Language

[ ] Svenska/Swedish
[X] Engelska/English
[ ] _____

Rapporttyp
Report category

[X] Licentiatavhandling
[ ] Examensarbete
[ ] C-uppsats
[ ] D-uppsats
[ ] Övrig rapport

_____

ISBN 91-7373-050-5

ISRN LiU-Tek-Lic-2001:27

Serietitel och serienummer
Title of series, numbering

ISSN 0280-7971

Linköping Studies in Science and Technology

Thesis No. 890

URL för elektronisk version

Titel
Title

Domain Knowledge Management in Information-providing Dialogue Systems

Författare
Author

Annika Flycht-Eriksson

Sammanfattning
Abstract

In this thesis a new concept called **domain knowledge management** for information-providing dialogue systems is introduced. Domain knowledge management includes issues related to representation and use of domain knowledge as well as access of background information sources, issues that previously have been incorporated in dialogue management.

The work on domain knowledge management reported in this thesis can be divided in two parts. On a general theoretical level, knowledge sources and models used for dialogue management, including domain knowledge management, are studied and related to the capabilities they support. On a more practical level, domain knowledge management is examined in the contexts of a dialogue system framework and a specific instance of this framework, the ÖTRAF system. In this system domain knowledge management is implemented in a separate module, a **Domain Knowledge Manager**.

The use of a specialised Domain Knowledge Manager has a number of advantages. The first is that dialogue management becomes more focused as it only has to consider dialogue phenomena, while domain-specific reasoning is handled by the Domain Knowledge Manager. Secondly, porting of a system to new domains is facilitated since domain-related issues are separated out in specialised domain knowledge sources. The third advantage with a separate module for domain knowledge management is that domain knowledge sources can be easily modified, exchanged, and reused.

Nyckelord
Keywords

Dialogue system, domain knowledge management, dialogue management, knowledge representation

Department of Computer and Information Science
Linköpings universitet

Linköping Studies in Science and Technology
Faculty of Arts and Sciences - Theses

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.

No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.

No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.

No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402	**Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406	**Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414	**Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agentteoretiskt perspektiv, 1994.
No 417	**Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436	**Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437	**Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440	**Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94	**Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94	**Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441	**Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446	**Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450	**Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451	**Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452	**Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455	**Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94	**Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462	**Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463	**Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464	**Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469	**Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473	**Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475	**Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476	**Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478	**Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95	**Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482	**Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488	**Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489	**Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497	**Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498	**Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503	**Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95	**Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95	**Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513	**Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517	**Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518	**Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522	**Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538	**Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545	**Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546	**Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96	**Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549	**Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550	**Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557	**Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558	**Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561	**Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563	**Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567	**Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575	**Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576	**Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587	**Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589	**Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591	**Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595	**Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597	**Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598 **Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hiera rchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16 **Marie-Therese Christiansson:** Inter-organistorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader,1999.
No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.
No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800 **Anders Subotic:** Software Quality Inspection, 1999.
No 807 **Svein Bergum**: Managerial communication in telework, 2000.

No 809	**Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32	**Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808	**Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820	**Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
No 823	**Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
No 832	**Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
FiF-a 34	**Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
No 842	**Magnus Kald:** The role of management control systems in strategic business units, 2000.
No 844	**Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
FiF-a 37	**Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.
FiF-a 40	**Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
FiF-a 41	**Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
No. 854	**Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
No 863	**Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
No 881	**Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
No 882	**Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
No 890	**Annika Flycht-Eriksson:** Domain Knowledge Management in Information-providing Dialogue Systems, 2001.