

Linköping Studies in Science and Technology

Thesis No. 1248

# The Use of Case-Based Reasoning in a Human-Robot Dialog System

by

**Karolina Eliasson**



Submitted to Linköping Institute of Technology at Linköping University in partial  
fulfilment of the requirements for degree of Licentiate of Engineering

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

Linköping 2006



# The Use of Case-Based Reasoning in a Human-Robot Dialog System

by

Karolina Eliasson

June 2006

ISBN 91-85523-78-X

Linköping Studies in Science and Technology

Thesis No. 1248

ISSN 0280-7971

LiU-Tek-Lic-2006:29

## ABSTRACT

As long as there have been computers, one goal has been to be able to communicate with them using natural language. It has turned out to be very hard to implement a dialog system that performs as well as a human being in an unrestricted domain, hence most dialog systems today work in small, restricted domains where the permitted dialog is fully controlled by the system.

In this thesis we present two dialog systems for communicating with an autonomous agent:

The first system, the WITAS RDE, focuses on constructing a simple and failsafe dialog system including a graphical user interface with multimodality features, a dialog manager, a simulator, and development infrastructures that provides the services that are needed for the development, demonstration, and validation of the dialog system. The system has been tested during an actual flight connected to an unmanned aerial vehicle.

The second system, CEDERIC, is a successor of the dialog manager in the WITAS RDE. It is equipped with a built-in machine learning algorithm to be able to learn new phrases and dialogs over time using past experiences, hence the dialog is not necessarily fully controlled by the system. It also includes a discourse model to be able to keep track of the dialog history and topics, to resolve references and maintain subdialogs. CEDERIC has been evaluated through simulation tests and user tests with good results.

*This work has been supported by the Wallenberg Foundation and the Swedish National Graduate School for Computer Science (CUGS).*

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



## Acknowledgements

First of all I would like to thank my supervisor Erik Sandewall for giving me free hands and believing in me. Without your gentle support and open mind this work would not have been possible.

I would also like to thank all members of the Cognitive Autonomous Systems Laboratory, both present members and past. I am especially grateful to my dear friends Peter Andersson and of course Malin Alzén and Susanna Monemar. Life in the laboratory is not the same without you! Tobias Nurmiraanta, thank you for still keeping me company and for the valuable comments on this thesis.

A special thank you goes to Daniel Bergström who has inspired me with crazy and creative thoughts and, most important of all, supported me lovingly in times of low confidence and disbelief. Thank you.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Research Challenges . . . . .	2
1.3	Research Contributions . . . . .	3
1.4	Publications . . . . .	5
1.5	Thesis Outline . . . . .	5
<b>2</b>	<b>Language and Dialog</b>	<b>7</b>
2.1	The Dream of the Talking Machine . . . . .	7
2.2	Dialog Manager Foundations . . . . .	8
2.3	Syntactic Models . . . . .	9
2.3.1	Syntactic Parsing . . . . .	9
2.3.2	Pattern Matching . . . . .	10
2.4	Dialog Models . . . . .	11
2.4.1	Dialog Grammars . . . . .	11
2.4.2	Plan-Based Models of Dialog . . . . .	11
2.4.3	Learning Dialog Policies . . . . .	12
2.5	Dialog Histories . . . . .	13
2.6	Domain Models . . . . .	14
2.7	Dialog with Robots . . . . .	14
<b>3</b>	<b>Planning and Learning</b>	<b>17</b>
3.1	Problem Solving versus Planning . . . . .	17
3.2	Learning . . . . .	18

---

3.3	Case-Based Reasoning . . . . .	19
3.3.1	Case-Based Planning . . . . .	21
3.3.2	Conversational Case-Based Reasoning . . . . .	22
<b>4</b>	<b>State of the Art</b>	<b>25</b>
4.1	The WITAS-Stanford Dialog System . . . . .	25
4.1.1	The WITAS UAV System . . . . .	25
4.1.2	The Dialog System . . . . .	29
4.2	The SmartKom Project . . . . .	31
4.2.1	The Discourse Model . . . . .	31
4.3	Robotic Dialog Systems . . . . .	34
4.3.1	Shakey . . . . .	34
4.3.2	KAMRO . . . . .	35
4.3.3	Jijo-2 . . . . .	36
4.3.4	Godot . . . . .	37
4.3.5	Carl . . . . .	38
4.4	Planning in Dialog Systems . . . . .	39
4.4.1	TRIPS . . . . .	39
4.5	Case-Based Reasoning and Dialog . . . . .	40
4.5.1	SiN . . . . .	41
4.5.2	The Discourse Goal Stack Model . . . . .	42
4.5.3	Case-Based Dialog Systems . . . . .	43
<b>5</b>	<b>Phase I - The WITAS RDE</b>	<b>47</b>
5.1	Background . . . . .	47
5.2	The WITAS RDE Architecture . . . . .	47
5.3	The Autonomous Operator's Assistant . . . . .	48
5.3.1	The SGUI . . . . .	49
5.3.2	The DOSAR Dialog Manager . . . . .	53
5.4	Simulations and Test Flights . . . . .	59
<b>6</b>	<b>The Research Problem for phase II</b>	<b>63</b>
6.1	Ideas Behind the Research Problem . . . . .	63
6.2	Problem Formulation . . . . .	64



---

<b>7</b>	<b>Phase II - CEDERIC</b>	<b>67</b>
7.1	Introduction to CEDERIC . . . . .	67
7.2	Design Choices . . . . .	70
7.2.1	The CBR Architecture . . . . .	70
7.2.2	Case-Based Planning . . . . .	71
7.2.3	Syntactic Model . . . . .	72
7.2.4	Dialog Model . . . . .	73
7.2.5	Dialog History . . . . .	73
7.2.6	Domain Model . . . . .	74
7.3	Discourse Model . . . . .	74
7.4	The Case Base . . . . .	77
7.4.1	Plan Case . . . . .	78
7.4.2	Plan Item . . . . .	80
7.5	Case-Base Manager . . . . .	83
7.5.1	Dialog Handling . . . . .	83
7.5.2	Syntactic Categorization of Words . . . . .	84
7.5.3	Case Retrieval . . . . .	86
7.5.4	Case Reuse . . . . .	87
7.5.5	Replanning . . . . .	91
7.5.6	Case Retention . . . . .	94
7.6	Learning from Explanation . . . . .	96
7.7	An Example . . . . .	97
<b>8</b>	<b>Tests and Results</b>	<b>103</b>
8.1	Development Tests . . . . .	103
8.2	Simulation Tests . . . . .	105
8.2.1	Scenario I . . . . .	107
8.2.2	Scenario II . . . . .	109
8.2.3	Results . . . . .	110
8.3	User Tests . . . . .	113
8.3.1	Results . . . . .	115
<b>9</b>	<b>Conclusion</b>	<b>117</b>
9.1	Retrospective . . . . .	117
9.1.1	Predecessors within WITAS . . . . .	117
9.1.2	Robot Dialog . . . . .	118

9.1.3 Case-Based Reasoning and Planning . . . . .	118
9.2 Main Results . . . . .	119
9.3 Future Work . . . . .	121
<b>Bibliography</b>	<b>123</b>

# Chapter 1

## Introduction

### 1.1 Motivation

Imagine that you are the organizer of a rescue mission in an area where a gas leak has occurred. The gas is poisonous and you do not want to go too close to the leak. To your help you have an autonomous unmanned aerial vehicle (UAV). The UAV is autonomous to that extent that it can perform missions and help you plan good moves. With your and the UAV's joint knowledge, you can search the area for people and trace the leak. You are using your voice and a headset to communicate with the UAV while you are busy working in the field yourself. It is not the first time you are operating it and the dialog system that controls the UAV has adapted to your use of language, and you hardly have to explain new words to it anymore. The dialog between you and the UAV runs smoothly and you can concentrate on the task at hand and do not have to use your mental resources on making yourself understood. Once in a while however, the UAV misinterpret or does not understand the phrase you communicate to it. When this happens the system performs a guess built upon the information from the context, previous dialogs and the words it understood. It presents the guess to you

and asks you for confirmation. If the guess is correct, you confirm and the dialog continues, and if it was wrong, you can correct it.

The benefit of a dialog system as the one just described is bigger than the obvious convenience in that particular scenario. It is very convenient to be able to control and interact with a system using natural language, because it is the main way of communication for people. The system is adapted to the preferred way of communication and the user does not need to learn an artificial language. If a system can understand a large number of phrases and dialogs, is able to learn from explanation, can adapt to its user, and is able to use both information from the context and from the current phrase at hand for interpreting the meaning of a phrase and react to it, then it is very adaptable to different tasks. It could be used for communication with different robots in different situations such as vehicles, kitchen machines and industrial robots, for communication with an autonomous travel agent via a telephone or for controlling your personal computer.

The system in the scenario is still far from reality, but it serves as an inspiration and guideline for further research in the area. This thesis will present two systems which aim at taking one step closer to the scenario.

## 1.2 Research Challenges

One major issue regarding dialog systems is how to specify what input the system understands and how it should react to it. Writing by hand a big system that can interpret a large number of phrases is tedious and time consuming. Further more, it demands a lot of knowledge about the domain, and case studies may be necessary to cover the whole spectrum of dialogs that may occur. Even if the system is well written and covers the current aspects of use, it is still static and can not adapt to a new situation. It is desirable to build a system that is flexible and adaptive to its environment.

The dialog system should also be able to understand and participate in a natural dialog. That includes the ability to solve references to items mentioned earlier in the dialog and to keep track of different subdialogs within a dialog. It should also understand the main topic and goal of the dialog to be able to react correctly. In addition to natural language,

human operators often tend to use gestures, such as pointing on a location on a map, when this feature is provided. A system that supports this multimodal communication is desirable, specially when the operator may want to talk about geographical information.

The following research challenges have been identified:

- The system needs to maintain a dialog history and to recognize the topic and goal of the dialog.
- The system needs some kind of machine learning technique which can increase the knowledge according to new experience over time. Which machine learning technique is best suited and how can it be integrated in a dialog system?
- To make effective use of the information in the system, it has to be able to reuse and combine the information in different ways and in various contexts. How can this best be done?
- Even if a good machine learning technique is implemented in the system, it will not be able to handle every possible input. One solution to this problem is to give the user the possibility to teach the system new concepts and dialogs at runtime, adapting the system to the user's use of language and the task at hand. How can this be done in practice?
- How can spoken natural language and other communication acts such as pointing gestures be integrated to a dialog system?

If these questions can be solved, we are one step closer to a dialog system that is more useful and more easily implemented and maintained.

### 1.3 Research Contributions

We have constructed two different dialog systems within the area of communication and control of an unmanned aerial vehicle. The first system called the WITAS RDE, focuses mainly on the first and the last research challenge identified in the previous section. It is an attempt to construct a

dialog system that is not more complicated than needed. The WITAS RDE is, despite its simplicity, an ambitious project which includes a graphical user interface with multimodality features, a dialog manager, a simulator and development infrastructures that provides the services that are needed for the development, demonstration, and validation of the dialog system.

The second dialog system is called CEDERIC. It is based on the dialog manager in the WITAS RDE, and focuses mainly on learning and discourse handling. It addresses all but the last research challenges listed in the previous section. The research contributions in the system can be summarized as follows:

- Different machine learning algorithms have been investigated and Case-Based Reasoning (CBR) has been chosen and integrated into CEDERIC. CBR works both as a design architecture and as a tool for adapting the system to new unseen phrases. The result is a dialog system that is able to deal with phrases that have not been stored beforehand in the system. The new knowledge is stored and the system performance improves over time.
- A discourse model has been integrated into the system, that keeps track of the dialog history and topics, solves references and maintains subdialogs.
- Case-Based Planning (CBP) has been investigated and implemented in CEDERIC, which makes CEDERIC handle dialogs that it has never seen before. By combining other dialogs using CBP, CEDERIC can increase its capacity automatically, with no human involved in the process.
- By specifying dialogs that guide the user through a learning phase where the system asks questions about an unknown word or a phrase, the system can learn from explanation and save the new information. This makes the system adaptable to different person's use of language and to the task at hand.
- CEDERIC has been evaluated through several tests and has been shown to work satisfactorily and to indeed implement the behavior described in the items above.

## 1.4 Publications

Parts of this thesis and predecessor versions of the CEDERIC system have previously been published as follows:

- [26] Karolina Eliasson. An Integrated Discourse Model for a Case-Based Reasoning Dialogue System. SAIS-SSLS event on Artificial Intelligence and Learning Systems, 2005.
- [27] Karolina Eliasson. Integrating a Discourse Model with a Learning Case-Based Reasoning System. DIALOR-05: the 9th Workshop on the Semantics and Pragmatics of Dialogue, 2005.
- [28] Karolina Eliasson. Towards a Robotic Dialogue System with Learning and Planning Capabilities. IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, 2005.

## 1.5 Thesis Outline

This thesis can be seen to consist of three parts:

The first part, consisting of chapter 2-4, provides background information about dialog systems, machine learning, and planning. Chapter 2 gives an overview of the concepts and techniques in the dialog system area. It explains the different parts that constitute a dialog manager and some different techniques for implementing the parts. Chapter 3 describes the foundations of machine learning and planning, in particular Case-Based Reasoning and Case-Based Planning. Systems that have served as an inspiration within the dialog manager area and the machine learning area are described in chapter 4. The WITAS UAV system, which includes the platform and robot that our dialog systems are designed to connect to, is presented in chapter 4 as well.

The second part describes the work that underlie this thesis. It has been divided into two separate phases; phase I and phase II. In phase I the author participated in the group that developed the WITAS RDE dialog system. The experiences from that system influenced the work with a successor of the WITAS RDE called CEDERIC, which is based on the WITAS RDE

but aims at somewhat different goals. The work with CEDERIC is called phase II. The WITAS RDE is described in chapter 5. Chapter 6 states the research problem for phase II and chapter 7 describes CEDERIC and phase II in detail.

The third and last part, consisting of chapters 8 and 9, presents tests, results and conclusions. Chapter 8 describes several different tests of CEDERIC and states the results in addition to comments about the performance of the system. Chapter 9 concludes the thesis with a retrospective section that compares CEDERIC with the systems described in chapter 4, a section that states the main results and, a final section that suggests areas of future work.



## Chapter 2

# Language and Dialog

### 2.1 The Dream of the Talking Machine

Ever since the dawn of the computer era, there has been a dream of creating a machine that is as intelligent as a human being. It was thought that the most obvious and natural way for the computer to manifest its intelligence was using natural language, either text or speech. Alan Turing presented his famous test known as the Turing Test in year 1950 [61]. The intention of the test is to validate if a machine can pass for being a human being. If so, the machine has accomplished true intelligence. When the test is performed, a person is placed behind a computer screen. She can write questions in natural language on the keyboard and answers appear on the screen. The conversation goes on for a while and after finishing the person has to guess if she was talking to a man or a machine. If she was talking to a machine but guesses a man, the machine is considered intelligent. That is, intelligence is measured by the performance of the interaction in natural language. Turing predicted that in year 2000 there would be systems that could fool the test person. Unfortunately he was wrong and no system has yet succeeded in the Turing Test.

Historically, two different research directions concerning dialog systems can be identified. The first one tries to develop a theory of dialog that

mimics human dialog as much as possible. Much work is done in cooperation with the human natural language community. Several logics have been proposed as a formal language for dialog modeling and complex use of dialog including different kinds of references and complex structures have been given lot of attention [2], [33].

The second direction addresses the implementation of dialog systems that can participate in a human dialog. These systems have traditionally been simpler than the dialog theory, and the performed dialog has little in common with dialog spoken between humans. These systems are typically database question-answering systems [15] or frame-filling systems [18]. The dialog in the former consists of the user asking questions to a database and the latter performs dialog by asking the user for information and filling in the information gathered in a frame. When all the slots have a value, the system can search for an item that matches the user's requests. This technique has been useful in, for example, travel booking systems [17], [53].

In the middle of the nineties, the speech recognizers and speech generators became better and dialog systems using spoken language became a popular research area, both in academia and in industry. This upswing in the interest of dialog systems has led to better integrated systems where ideas from the dialog theory have been integrated into actual performing dialog systems [10].

## 2.2 Dialog Manager Foundations

A simple and common architecture for a spoken dialog system is shown in Figure 2.1. The speech recognizer translates from speech to text and returns the best match in text format. The dialog manager interprets the meaning of the input and creates a suitable response. The response is sent to a speech generator which articulates the response to the user. The speech recognizer and the speech generator are often off-the-shelf products, but in some systems they are integrated with the dialog manager. In a system where, e.g., the speech recognizer is integrated with the dialog manager, information about previous utterances and expectations of the continuation of the dialog, can be used to recognize and interpret the user utterances. The disadvantage is a more complex dialog manager. This the-

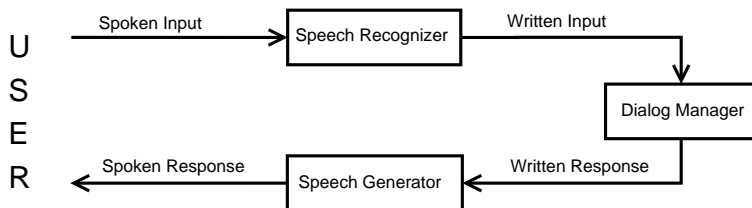


Figure 2.1: Architecture of a typical spoken dialog system.

sis is, however, mainly focused on a stand alone dialog manager as shown in Figure 2.1.

The dialog manager is the engine of the dialog system. It interprets the meaning or intention of the input from the user. The input can be a continuation of an earlier input or refer to something mentioned earlier in the dialog. It is the task of the dialog manager to solve references, unwind the information behind the input and act on it. An action could be to ask a clarifying question, to perform a database search and return the result, or to send a command to the control system of a robot.

## 2.3 Syntactic Models

When the dialog manager receives the input, it first has to recognize the words in the input. Without any knowledge about the words in the input, the system can not interpret the meaning further. There are several methods for doing this, in particular syntactic parsing [4] and pattern matching. Combinations of these and integration with the more semantic steps further on in the work sequence occurs as well [21].

### 2.3.1 Syntactic Parsing

Parsing is the process of analyzing and recovering the syntactic structure of a sentence given a grammar. The grammar consists of a lexicon and a rule set. The lexicon lists all the allowed words and categorizes them into

groups such as nouns, verbs etc. Each word in the sentence is looked up in the lexicon and tagged with the corresponding class. The tagged words are then grouped together into phrases. A phrase could be, e.g., a noun phrase or a verb phrase. The rule set indicates which classes of words have to be present to form a phrase and in which order they should appear in the sentence. The parser returns a parse tree if the sentence was syntactically correct according to the grammar. The parse tree can then be used to make a semantic analysis of the sentence.

Syntactic parsing gives an exact analysis of the syntactic structure which can be very useful when the system has to distinguish between sentences that are similar. One drawback is that all sentences which the system should be able to interpret must be specified in the grammar. It is well known to be both time consuming and difficult to cover all possible utterances that the user may use when interacting with the system.

### 2.3.2 Pattern Matching

When pattern matching is used, the input sentence only needs to contain some keywords to be considered a match. The system is equipped with scripts that contain a sequence or a set of keywords. The sentence is searched for the keywords and if all keywords are represented and no other script matches the sentence better the sentence is considered a match and a response can be generated automatically or information coupled to the script can be used for further semantic analysis.

This method has some appealing advantages. It is easy to implement and intuitively understandable. It can deal with a large range of different input sentences, using one and the same script, hence it is no need to specify all variants of accepted input on a more linguistic level. Adding new scripts is easy and no special competence is needed by the software developer. Unfortunately, it also has some drawbacks. It can not distinguish between similar sentences with different meanings if the distinguishing word is not a keyword. It also has problems if references are used that refer to a keyword mentioned earlier in the dialog. Another drawback is that it does not create detailed information about the composition of the sentence to be used in the further steps of the dialog manager.

## 2.4 Dialog Models

For a dialog system to be able to participate satisfactorily in a dialog, it has to be able to recognize and analyze different dialogs. Human dialogs can be characterized by a sequence of speech acts, where a speech act is a classification of the utterance. It can for example be a request, a question or a reply. The dialog model of a dialog system works as a guide and helps the system to characterize the input from the user and to perform an appropriate response to the input.

### 2.4.1 Dialog Grammars

It has been suggested that a dialog can be represented theoretically by a grammar in the same manner as a sentence can be represented in syntactic parsing. The foundation of dialog grammars is the notion of adjacency pairs which is built on the observation of regularities in a natural dialog, e.g., that a question often is followed by an answer in a well defined dialog. The question and the answer form an adjacency pair. The rules in a dialog grammar state which communicative actions can form adjacency pairs and hence states the sequential and hierarchical constraints on the dialog. An adjacency pair or a number of adjacency pairs can be grouped together to model the stage of the dialog so far, e.g., initiating and reacting stages. A dialog can then be parsed using the grammar and a set of suitable system responses can be found according to the past dialog.

Dialog grammars are easy to implement and to understand. The method works well in small systems where the dialog is controlled and restricted. It is however not suitable in a system where the user may merge several communicative units into one utterance. Another severe drawback is that the method does not include a rule for choosing between several possible responses [23].

### 2.4.2 Plan-Based Models of Dialog

An essential concept in plan-based models of dialog [11], [29], [36] is the notion of dialog acts. A dialog act is a speech act that occurs in a dialog representing the intentional meaning behind an utterance, e.g., requesting,

suggesting or confirming. An utterance from a speaker is more than just a sequence of words, it is an action performed by the speaker to achieve a goal. Several utterances in a dialog may be needed to achieve the goal and the speaker plans the dialog acts according to the goal. A goal may be to make the listener perform an action or to update the mental state of the listener. The listener's part of the dialog is to uncover the plan with the corresponding goal and to respond appropriately to it. With this plan-based approach, dialog acts can be seen as a special case of other actions and action plans, and planning becomes an important issue. Planning is a major area of research in the AI field and a lot of work can be found in the literature. More information about planning can be found in the next chapter. Other aspects of plan-based models is how the listener is affected by the speaker's plan or intentions and how they can carry out the dialog with joint effort.

The advantage that the dialog can be treated as a special case of action planning has some not so desirable side effects, namely the necessity of a distinction between task-related speech acts and those used to control the dialog, such as clarifications. Another drawback of plan-based models of dialog is the lack of a theoretical foundation at present. The method is purely procedural and the definitions of, e.g., plans and goals are vague [23].

### 2.4.3 Learning Dialog Policies

Modeling dialog by hand, as is necessary in dialog grammars and plan-based models of dialog, is difficult and demands rigorous tests and feasibility studies. One approach to overcome these problems is to use machine learning techniques to learn the best dialog strategy, as described in [40]. The machine learning technique most often used is Markov Decision Processes (MDP) and a corresponding reinforcement learning algorithm to estimate the optimal strategy [3]. When modeling the dialog system as an MDP problem one has to describe it as a sequential decision process in terms of its action set, state space and strategy. The action set consists of all the actions the system can perform, e.g., communication actions or database lookups. The state space is all states the system can be in. A state includes the values of all the variables that determine the next action. The strategy defines, for each state, which is the optimal next action to per-

form. By using a reinforcement learning algorithm, the strategy, or policy, that is the most optimal for each state can be learnt from examples and cost/reward measurements. The learning phase demands a large number of test dialogs, that can not be directly taken from a corpus because the system may give a different response to an input than what is specified by the corpus dialog. This can be solved by using a human test person but the testing is time consuming for a human to perform. The general solution is to build a simulated user who can interact with the system during the test phase.

The problem increases rapidly with the size of the action set, and a bigger action set demands a longer learning phase with more complex learning examples. Therefore, a handcrafted policy is often used when no ambiguity is present. Learning dialog policies has been shown to be a good method to find out which strategy to choose when several different dialog acts can be performed. The problem can, e.g., be to choose between a phrase that asks for several values at the same time, such as a date, or to ask one restricted question for every value, such as asking for the day and month separately. Some work in the field can be found in [32], [40], [58], [62]. Besides scalability problems, the approach also has some drawbacks regarding dialog phenomena such as references, and it does not provide any guidelines about what information to include in a state and what possible actions a domain may have.

## 2.5 Dialog Histories

A dialog history or discourse model is a model of the contents of the dialog up to a given point. It is used mainly to solve references to objects mentioned in earlier utterances, but could also be useful to keep track of different ongoing dialogs with different topics. A dialog history can vary in complexity and level of detail. A simple solution is to save all the objects mentioned in the dialog on a stack, sometimes called a salience list, and assume that all references refers to the last mentioned object. A more sophisticated solution is to save the dialog histories as a tree, where a new dialog is started when the topic changes. The dialog tree can have several levels of information, e.g., the dialog level which models the overall topic of

the dialog, the speech act level which models the information in the speech act and the object level which models the objects mentioned in the dialog and their attributes [52]. The different levels give information about the whole dialog, which utterances that are semantically close together because they handle the same topic and more detailed information about words that may be referred to later on in the dialog.

## 2.6 Domain Models

A domain model consists of the domain knowledge of the world that is described in the system's input and where the system is required to react in a proper manner. The knowledge is often used to connect the natural language to the back end system, e.g., a database or a robotic control system. There are no well defined general domain models in the dialog literature, and the domain models in actual systems range from non existing to full-fledged world models with reasoning capabilities. However, some general ideas can be seen in various dialog systems. General knowledge about objects and relations in the world can be modeled using an ontology, represented in, e.g., XML. Specific knowledge can be stored in the lexicon of the grammar as in WAXHOLM [21].

## 2.7 Dialog with Robots

Using natural language for interacting and controlling a moving robot demands some additional considerations regarding the design of the dialog system. An important aspect is robustness of the system, due to the real world application of the robot [35]. Another aspect to consider is the dynamic environment, which may lead to the necessity of a large number of different types of dialogs and may require a flexible dialog system that can deal with rapidly changing dialog topics [37]. A third issue is the adaptation to the ever changing dynamic environment. Some learning strategies have been implemented in ground robots such as Carl [41] and Jijo-2 [16], see section 4.3 for a more detailed description of the systems.

A big issue in robotic dialog systems is how to integrate multimodal-



ity into the systems [37]. Multimodality is the concept of using several communication medias, such as gestures on a touch screen or selection of items from a menu in collaboration with natural spoken language. In many applications, pointing on an item or spot to select it is a very natural way of communication, and a system that handles speech alone seems too complicated in such situations.



# Chapter 3

## Planning and Learning

### 3.1 Problem Solving versus Planning

Problem solving and planning are two related concepts for reasoning in an agent. By problem solving one often means searching for a solution to a problem in a search space consisting of situations, using a search algorithm such as breadth first or A\*. Planning can be viewed as a type of problem solving in which the agent uses beliefs about actions and their consequences to search for a solution. Problem solving and planning are similar in the sense that they both start with an initial state and try to find a solution to some problem. Problem solving can be performed in several different manners, e.g., a search in the search space from the initial state to the goal state, possibly using a heuristic function. The path from the initial state to the goal state is the solution to the problem. Problem solving can also be performed using reasoning techniques such as logic or rule based systems. Planning is a special form of search based problem solving. The main idea is similar where the solution is a plan from the initial state to the goal state. However, in search based problem solving, the solution is built up incrementally from the initial state to the goal state, which can be very resource consuming in large search spaces. In planning, the planner is free to add actions to the plan wherever they are needed. Several subplans can

be constructed separately and merged at a later time. Another difference is the representation of states, goals, and actions. In problem solving, they are considered as "black boxes", but in planning, they are defined by some formal language, where an action usually consists of an action description, preconditions, and effects. A well known formal language for planning, which has served as a foundation for several other languages, is the STRIPS language [30]. An introduction with references to problem solving and planning can be found in [4].

## 3.2 Learning

Machine learning techniques enable a program to improve automatically with experience. We do not yet know how to design computers that learn as well as people do, but in certain types of areas there exist fairly efficient learning algorithms. Speech recognition is an example of an area where machine learning techniques outperform all other approaches that have been attempted [6].

Most learning algorithms need a training phase during which the system is given a number of training examples. Often the training examples consist of a problem formulation and a solution to the problem. The task of the learning algorithm is to induce general functions from these specific training examples. When the learning phase is completed, the general functions obtained are fixed and new problems are evaluated using these functions. No new information can be learnt after the training phase. However, since the machine learning area is broad and interdisciplinary, and draws on concepts from many fields, including statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory, the various proposed algorithms differ significantly from each other. Well known machine learning algorithms are decision-tree learning, artificial neural networks, rule-based learning and bayesian learning. See the textbook *Machine Learning* by Tom Mitchell [6] for an introduction to the area.

### 3.3 Case-Based Reasoning

Case-Based Reasoning (CBR) is a machine learning method and a problem solving algorithm used both for standard classification problems and other learning and problem solving methods where the knowledge base is increased over time. Using CBR, a system is able to learn from experience even when the training phase is completed because it uses specific information instead of general knowledge as usually used in other machine learning methods. In this way it does not have to construct a set of general functions from the specific examples, but stores the specific cases as they are in a case base. The actual learning computations are performed for every new problem that enters the system. This is called lazy learning, in contrast to eager learning as used by most other learning algorithms.

The method originates from studies of dynamic memory [1] and is greatly influenced by cognitive psychology. The main idea is to solve a new problem by remembering a similar situation and reusing information from that situation. The information or knowledge gathered is stored in the case base. Each case in the case base has a problem part and a corresponding solution part.

The CBR cycle described by Agnar Aamodt in [8] and shown in Figure 3.1 consists of four processes:

- *Retrieve* the most similar case or cases.
- *Reuse* the information and knowledge in that case to solve the problem.
- *Revise* the proposed solution.
- *Retain* the parts of this experience likely to be useful for future problem solving.

A description of a new problem enters the system and constitutes a new case. The information in the new case is used to *retrieve* an old case from the case-base, which is similar to the new case. The retrieved case is combined with the new case through *reuse* into a solved case, i.e. a proposed solution to the new problem. In the *revise* process the new solution is tested by evaluation. If it fails, it is repaired, and an opportunity

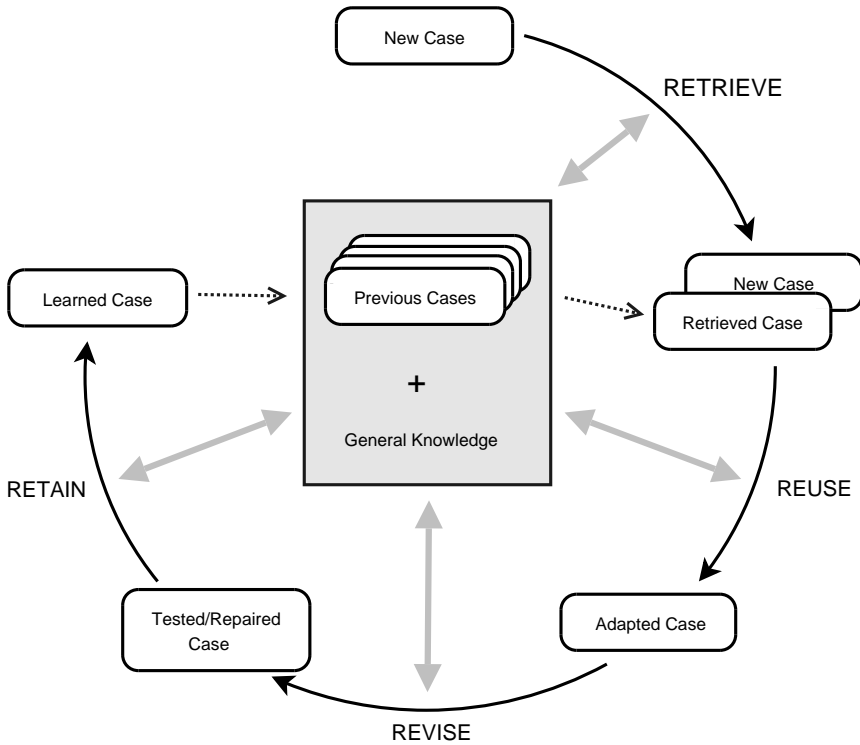


Figure 3.1: The CBR cycle.

for learning from failure arises. In the retaining process, useful experience is *retained* for future reuse, by saving it in the case base.

The two most problematic processes are the *retrieve* process, where the most similar or suiting case in the case base is found, and the *reuse* process, where the solution in the similar case is combined and adapted to be able to solve the new problem.

In the *retrieve* process, the problem description is examined. Some descriptors may have a higher information gain according to the problem

than other. The sifting process may need some general knowledge about the domain in which the system operates. When the descriptors are ranked according to their importance, the search for a similar match can begin. Similarity can be based on superficial, syntactical similarities or on deeper, semantical similarities. Syntactical similarities can often be obtained by a simple equality function, but semantical similarities demand more sophisticated similarity functions. General knowledge may be used to guide the search and to assess the degree of similarity. Cases may be retrieved solely from input features, or also from features inferred from the input.

The *reuse* process can be performed in several different manners. In simple classification problems, the solution of the old case can be used directly as a solution to the new case, because if the cases are considered similar, they belong to the same class. If the old solution is not directly applicable, an adaptation process has to be performed. A solution can be adapted in two different ways:

- reuse the old case solution or
- reuse the old method that was used to construct the solution.

In the first approach, a set of operators have to be defined for the transformation. These operators transform the old solution to a solution that solves the new problem. One way to organize them is to index them around the differences between the old and the new case. The latter approach looks at how the problem was solved in the old case and adapts the method that constructed the solution. Information about the solution strategy and justifications for the methods used are parts of the solution of a problem, and can be adapted by the system.

### 3.3.1 Case-Based Planning

In Case-Based Planning (CBP) [43], [59], CBR is used as a planning method, not only as a problem solving method. A case is made up by the following parts:

- A *Problem Part* consisting of information about the initial state and the goal.

- A *Plan* which is a sequence of actions or decisions that solve the problem.

The similarity function does not only depend on the problem part in this case, but the plan of the found case should be easy to adapt to the new problem. When reusing an old plan, the decisions that were made and stored in the plan are replayed and some decisions can be reused while the remaining decisions are taken by replaying another case, or by using a generative planner.

### 3.3.2 Conversational Case-Based Reasoning

CBR has been used in various domains, and a subarea of CBR called Conversational Case-Based Reasoning (CCBR) has arisen [9]. It was the first widespread commercially successful form of CBR. The CBR methods described above rely on the user to provide complete knowledge about a problem, but this is not always possible in practice. In many domains, the user does not know all the features of the problem or does not know in which form she should provide them to the system. CCBR uses written dialog in natural language to query the user for more information about the problem. It also uses human-in-the-loop for finding similar cases, by showing the user a set of similar cases and letting the user choose the best one.

A case in CCBR consists of the following parts:

- A *Problem Part* consisting of a partial *description* of the problem and a *specification*, which is a set of <question, answer> pairs.
- A *Solution Part* which is a sequence of actions responding to the problem.

The user provides the system with the initial description of the problem in written natural language. The system uses pattern matching, see section 2.3, to extract feature values from the input. These features and values constitute the initial problem. The system compares the initial problem formulation with the problem part of the cases in the case base and rank them according to similarity. The solution to the most similar cases



are displayed in a graphical representation to the user, together with a selection of questions which ask for values of distinguishing features. The answers to the questions can be used to further distinguish the cases and to add information to the problem formulation. The user can choose either to directly select a solution which ends the interaction sequence, or to answer one of the questions. If a question is selected and answered, the feature and the answer are added to the problem formulation, and the system starts to search for similar cases again. The system loops until the user selects a solution or until there is only one possible solution. It can also detect if there is no solution to the particular problem in which case it will inform the user.

The major difficulties when using CCB<sub>R</sub> are how to rank the cases by similarity and how to rank the questions by importance and capability to distinguish between similar cases.

The dialog in CCB<sub>R</sub> is in a simple question-answer form. Current CCB<sub>R</sub> systems do not provide any support for references or subdialogs.



# Chapter 4

## State of the Art

### 4.1 The WITAS-Stanford Dialog System

The main goal of the WITAS project [24] is to develop an integrated hardware/software VTOL (Vertical Take-Off and Landing) platform for fully autonomous missions and its deployment in applications such as traffic monitoring and surveillance, emergency services assistance, photogrammetry and surveying. The project is an umbrella for several multi-disciplinary research areas such as UAV control, robot architectures [25], planning [51], image processing [50], and dialog systems for support of ground operation personnel. Several dialog systems have been developed within the project, each of them with different strengths and weaknesses. The dialog systems are presented in section 4.1.2, chapter 5 and chapter 7.

#### 4.1.1 The WITAS UAV System

The particular aspect of WITAS that is relevant here is the dialog system. However we provide an overview of the WITAS UAV system background.

The hardware currently used is a modified Yamaha RMAX radio controlled helicopter, shown in Figure 4.1. It has a total length of 3.6 m and a maximum take-off weight of 95 kg. A video camera and three embedded



Figure 4.1: The WITAS Yamaha RMAX helicopter.

computers are mounted on the helicopter along with a number of sensors including a GPS, a compass, and a barometric altitude sensor.

Several different control modes including a high-level interface to the control system are implemented in the agent architecture [25]. The following flight modes has been implemented and tested:

- autonomous take-off and landing via visual navigation
- autonomous hovering
- autonomous dynamic path following

- autonomous reactive flight modes for interception and tracking.

A task procedure (TP) is a computational mechanism which implements a behavior intended to achieve a goal in a limited set of circumstances. A TP can be called, spawned or terminated by an outside agent or by other TPs. In our system, a TP can, e.g., call a path planning service or call one of the flight modes described above.

The architecture contains two information repositories:

*The Knowledge Structure Repository.* This repository contains high-level deliberative services such as the task planner and the chronicle recognition packages. It also includes the Dynamic Object Repository (DOR). The DOR is a database containing moving objects recognized by the vision system.

*The Geographic Data Repository.* The static geographic data of the environment is stored in a Geographic Information System (GIS). It includes a highly accurate terrain model with decimeter precision in the x, y and z directions and equally accurate models of buildings and road structures. The GIS is used by the path planner [51] which is called with two waypoints and optional constraints as parameters and generates a collision free and smooth path between them.

System tests have been performed in Revinge in southern Sweden, where an emergency services training school is located. The area consists of buildings and roads and is well suited for realistic test mission flights. Several high-level autonomous flights have been performed such as tracking of a moving car and autonomous take-off and landing including a flight where the flight plan was created by the path planner.

Within the WITAS project, several dialog systems with various capabilities have been developed. The first WITAS Dialog System [37], [38], [39], [55], described in section 4.1.2, was a system for multi-threaded robot dialog using spoken I/O. The WITAS Robotic Dialog Environment (RDE) [55], [56], described in chapter 5, is a new implementation using another architecture and a logic base. CEDERIC is another dialog manager that partly use the same components as the WITAS RDE. It focuses on integrating machine learning features into a dialog system.

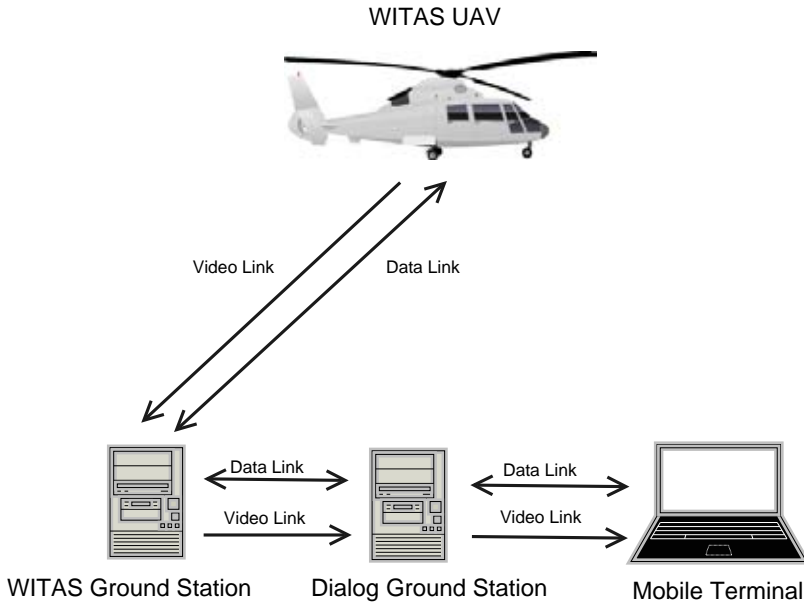


Figure 4.2: The WITAS system setup.

The various dialog systems have the same connection interface to the UAV. The UAV communicates with a ground station using three independent communication channels. The first one is a simple remote-controlled channel provided by the manufacturer. The second is a two-directional data link for downloading various data and uploading commands to the helicopter. The third is a video link sending a composite video stream from the onboard camera to the ground personnel. The dialog system consists of a mobile terminal running a user interface, and a dialog ground station running the dialog manager and a video server. When the operator issues a command to the helicopter it is sent from the mobile terminal to the dialog ground station, and further to the UAV ground station where it is checked by a human supervisor. If it passes the check it is sent to the

UAV onboard system for execution. The WITAS system setup is shown in Figure 4.2.

### 4.1.2 The Dialog System

The WITAS-Stanford Dialog System [37], [38], [39], [55], was developed by Lemon and Peters et. al. at Stanford University. It was the first dialog system created for the WITAS domain within the project. The system focuses on the following requirements:

- *Asynchronicity*: events in the dialog scenarios interesting to the dialog system can happen at overlapping time periods.
- *Mixed task-initiative*: both the operator and the system may start a new dialog thread.
- *Open-ended*: there are no clear start and end points for the dialog and subdialogs, nor are there rigid pre-determined goals for interaction.
- *Resource-bounded*: participant's actions must be generated in time enough to be effective dialog contributions.
- *Simultaneous*: participant can produce and receive actions simultaneously.

To meet these challenges, a flexible architecture which can coordinate multiple asynchronous communication processes, is needed. The dialog system is made up by several agents, connected using the Open Agent Architecture [42]. The most important agent is the dialog manager. It creates and updates an Information State corresponding to a notion of dialog context. Dialog moves performed by the robot or the operator have the effect of updating information states. An information state consists of the following structures:

- *Dialog move tree*: a tree that represents the structure of the dialog by way of conversational threads, composed of the dialog moves of both participants, and their relations.

- *Activity tree*: a tree that represents hierarchically decomposed tasks and plans of the robot, and their states.
- *System agenda*: a list of the planned dialog contributions of the system and their priorities.
- *Pending list*: a list that represents open questions raised in the dialog.
- *Salience List*: a priority-ordered list of the objects referenced in the dialog thus far. This list also keeps track of how the reference was made. It is used for identifying the referents of anaphoric and deictic expressions and for generation of contextually appropriate referring expressions.
- *Modality Buffer*: a buffer that keeps track of the mouse gestures until they are either bound to deictic expressions in the spoken output or, if none such exists, are recognized as purely gestural expressions.
- *Databases*: dynamic objects, planned routes, geographical information, and names.

The dialog manager is able to support commands, questions, revisions, and reports, over a dynamic environment. It performs mixed-initiative, open-ended dialogs and operates in an asynchronous, real-time, multimodal fashion. It also performs constraint negotiation and implements interleaved task planning and execution. A Graphical User Interface which allows deictic reference such as mouse pointing from the user and displays route plans, waypoints and locations of vehicles including the robot has been developed as part of the system.

An evaluation of the system was conducted, measuring task completion rates for novice users with no training. 55% of novice users were able to complete their first task successfully, rising to 80% by the fifth task.

The most notable difference between the WITAS-Stanford dialog system and other dialog systems is the absence of predefined plans or patterns due to the use of Information States which provides a flexible way to process the conversation.



## 4.2 The SmartKom Project

SmartKom [7] is a multimodal dialog system combining speech, gesture and facial expression input and output. The user can operate the system by using a combination of spoken language, gestures and facial expressions. The graphical user interface includes an animated life-like character called Smartakus, who presents the output using graphics of various kind, gestures and speech. The main goal of SmartKom is to support the access to knowledge-rich services, using spoken language, gesture, and facial expression in a coordinated and intuitive way. SmartKom defines three different scenarios where the dialog may be useful:

- *SmartKom-Public*: The public scenario describes a multimodal communication booth which provides access to information concerning hotels, restaurants, and entertainment.
- *SmartKom-Mobile*: The mobile scenario describes a PDA version providing a navigation system both for car drivers and pedestrians.
- *SmartKom-Home*: The home scenario describes a service portal that permits the controlling of, e.g., TV, VCR, telephone and e-mail, and gives access to various information sources such as TV guides.

One of the long term goals in SmartKom is to develop a reusable multimodal dialog backbone with a generic discourse model.

### 4.2.1 The Discourse Model

The dialog system in SmartKom does not use a conventional dialog manager as seen in other applications. Instead the dialog manager is split up into a discourse modeler and an action planner. The system is multimodal and both speech, gestures and facial expressions are taken into consideration. An input from the user is analyzed separately in various analyzers depending on modality. Each analyzer provides a hypothesis of the semantic meaning of the input as well as a confidence score. Hypotheses from the different analyzers are brought together in the modality fusion. The discourse modeler ranks and selects, based on the scoring from all analysis components, the most probable hypothesis which is then passed on to

the action planner. The action planner handles the input according to the hypothesis and passes presentation goals on to the presentation manager and publishes expectations regarding the following user input. The presentation manager plans the output depending on available and suitable modalities.

The system uses a rich ontology including processes and objects, which, e.g., gives a complete description of an action such as browsing a database.

The discourse model in the SmartKom project is responsible for the following main tasks:

- *The enrichment and validation of intention hypotheses*, i.e. determining the user's intention by validation and enrichment of the intention hypotheses from the analyzers. The enrichment is performed by using information from the preceding discourse.
- *The resolution of referring expressions*, i.e. the user has uttered a referring expression that is not accompanied by a deictic gesture.

Both tasks need access to a multimodal contextual representation of the preceding discourse.

The context representation of the discourse model is separated into three different layers:

- The Modality Layer.
- The Discourse Object Layer.
- The Domain Object Layer.

The Modality Layer consists of objects encapsulating information about the concrete realization of a referential object depending on the modality of representation. Corresponding to the three different types of modality, the modality layer has three different types of objects:

- *Linguistic Objects (LOs)*. For each occurrence of a referring expression in a generated or interpreted utterance one LO is added.
- *Visual Objects (VOs)*. For each visual presentation of an object that can be referred to one VO is added.

- *Gesture Objects (GOs)*. For each gesture performed either by the user of the system one GO is added.

Each modality object is linked to a corresponding discourse object.

The Discourse Object Layer is the most central structure in the context representation. It consists of objects that represent concepts participating in the communication and that can be referred to by other referring utterances. When the concept is introduced for the first time, a discourse object (DO) is created. Every DO is unique and if the concept is referred to, an object in the modality layer is created and linked to the DO. The discourse object layer can handle composite objects and references to objects within a composite object. A composite object is called a partition. A DO that contains a partition is created to model the composite object as a whole, e.g., a list structure. It has pointers to other DOs that represent the individual items within the object, in our example the items in the list structure. In this way, individual objects as well as the composite object as a whole can be referred to by the user. This structure is useful, e.g., when the system utters an enumeration of objects and the user chooses one of them by saying, e.g., **the second one**.

The Domain Object Layer provides the mapping between a DO and instances of the domain model. For each user or system intention at least one domain object exists which describes the intention. This representation is anchored in the structure of the dialog, which is represented in terms of turns. A turn starts when a speaker starts to speak and ends when she stops speaking and another speaker takes the turn by start speaking.

The different turns in a dialog are represented in a global focus stack. The global focus stack consists of global focus spaces, where each focus space covers the turns of the dialog participants belonging to the same topic. In addition, there is a local focus stack responsible for the access to the discourse objects. A local focus space contains pointers to discourse objects that are antecedent candidates for later reference. Every global focus space has a pointer to the corresponding local focus. Several different dialogs can be active at the same time in the global focus stack.

To group the turns in a dialog together topically, simplified, non-hierarchical initiative-response units are used. The initiative-response units restrict the access to possible referents and provide structure to the global focus stack.

The most recently used global focus space is on top of the stack and currently in focus. If an utterance enters the system and does not fit that focus space according to the initiative-response units and overall topic of the dialog, then other open global focus spaces are examined. If the utterance matched another focus space, it is moved to the top of the stack. In this way, several dialogs can be open and ongoing at the same time. A dialog is considered closed if the focus space only contains closed initiative-response units, i.e. every utterance has got a corresponding matching response utterance.

## 4.3 Robotic Dialog Systems

Robotic dialog in general, and some main questions at issue are presented in section 2.7. Even if dialog systems and robotics are two major areas in Artificial Intelligence, they have been studied rather independently in the past. However, some work has been done in the field of robotic dialog systems over the years. In this section we present some projects and robots, which have integrated a dialog system with a physical robot. Apart from the systems presented here, the WITAS-Stanford Dialog System presented in section 4.1.2 is an interesting example of a robotic dialog system.

The WITAS-Stanford Dialog System, Shakey, and KAMRO are examples of robots that integrate a dialog system as an interface to the robotic control system. Jijo-2, Godot, and Carl are examples of robot systems that integrate some element of learning from explanation using dialog in spoken natural language.

### 4.3.1 Shakey

The famous robot Shakey [49], was developed at the Artificial Intelligence Center at SRI during the years 1966 to 1972. Shakey was an embodied physical robot with two drive wheels, a video camera, a range finder sensor and bump sensors. It could perform tasks that required planning, route-finding, and rearranging of simple objects. Shakey was the first robot that could claim to reason about its actions. The planner used was an implementation of the STRIPS language mentioned in section 3.1. Shakey was

able to visually interpret its environment, to locate items, navigate around them, and to understand simple commands given in natural language.

### 4.3.2 KAMRO

KAMRO [35] is a robot developed within the VITRA project. It is a two-armed robot-system with sensors for navigation, docking, and manipulation. KAMRO works in the workbench domain where the two arms are used to grasp and move objects such as shafts, levers, and spacing-pieces. The main issue of the VITRA project is to bridge the gap between natural language and vision.

KAMRO does not use natural language only as a command language, but the robot can participate in a dialog with the user to resolve ambiguities and misunderstandings. KAMRO uses a dialog system called KANTRA. The following situations have been identified where natural language can be useful within the project.

- *Task specification:* The operator can give commands to the robot at several different levels of abstraction: from high-level commands like `assemble benchmark`, implicit robot operations, e.g., `pick side-plate`, to explicit robot operations like `grasp`.
- *Execution monitoring:* An autonomous system is able to plan an execution of a high-level command on its own. It can however be interesting for the operator to be informed about what the robot is actually doing.
- *Explanation of error recovering:* An autonomous system that is able to detect errors and recover from them may not behave as expected from the operator's point of view. The ability to explain how and why plans have been changed may increase cooperativeness.
- *Updating and describing the environment representation:* Since the visual field of an autonomous mobile is restricted, some information about the physical world may be missing in the robot's representation of the world. The operator can aid the robot in maintaining the world representation by providing additional information in natural

language. The other way around, the operator should also be able to ask for a verbal description of the scene.

KANTRA is able to perform a dialog in natural language in the above described situations. Spatial reasoning is used to be able to understand utterances such as `near the spacing-piece`.

### 4.3.3 Jijo-2

Jijo-2 [16] is an office-conversant robot, which serves as a platform for the research towards socially embedded learning. Socially embedded learning means that the system learns through a close interaction with its social environment, including one or several teachers.

The robot used in the project is a mobile robot which autonomously walks around in an office environment, actively gathers information by sensing multimodal data and engaging in dialog with people in the office, and acquires knowledge about the environment with which it ultimately becomes conversant. The main goal is not for the robot to learn lower level functions but to learn, e.g., topological and geometric information in the environment from a teacher in combination with own experiences.

The robot is equipped with various sensors such as ultrasonic sonar, a microphone, a camera, and a speech synthesizer. The speech signals recorded by the microphone are transmitted to a host computer via a radio transmitter. To be able to learn map information, the robot can ask questions and ask for guidance from a human teacher. A typical conversation can start with a command from the teacher, where she wants the robot to go to someone's office. If Jijo-2 does not know where the office is, the teacher can give commands such as `go straight` and guide the robot to the office. During the learning phase, the robot is building a map over the traveled area, using a partially observable Markov model. The teacher can also choose to say the command `follow me`, which makes the robot follow the teacher using visual tracking.

Jijo-2 has been tested in a real environment, and after 52 trial runs it succeeded to create a map consisting of 14 state nodes corresponding to different specific locations.

#### 4.3.4 Godot

Godot [60] is a small, cylindrical robot equipped with 16 sonar, infrared and collision sensors. It also has two wheel encoders and a camera mounted on a pan-tilt unit. As for Jijo-2, the main purpose of the project is to investigate the interface between a navigation system and a spoken dialog system. The information flows in two directions between the navigation system and the dialog system. The low-level navigation system supplies landmark information from the cognitive map used for the interpretation of the user's utterances in the dialog system. The high-level dialog system also influences the navigation system by the way the semantic content of utterances analyzed by the dialog system are used to adjust the probabilities about the robot's position in the navigation system.

The dialog considered in the project is of the following nature:

- The human informs the robot about its current position, possibly in response to a question from the robot after finding out that it is uncertain of its location.
- The human queries the robot about its current beliefs about its position, possibly followed by a correction or confirmation by the human.
- The human instructs the robot to move to a certain position.

Communication is performed in a natural, unrestricted spoken English. The human should be able to label places such as **the kitchen** and **my office** and these labels are not necessary unique. This not only has an impact on the design of the cognitive map, but also requires ontological knowledge and a semantic representation of the dialog which enables the robot to perform inference. The robot should also be able to understand utterances that is sensitive to context, hence the semantic formalism is able to deal with anaphoric and deictic references.

The cognitive map consists of three layers, a geometric layer, a topological layer, and a semantic layer. The geometric layer is a grid modeled as a 0-dimensional Markov random field. The topological layer is derived directly from the geometric layer, by dividing the free space into rooms or corridors. The semantic layer, labels the different areas in the topological

layer. The construction of the maps is learned using a multi-layer neural network.

The dialog component uses Discourse Representation Structures (DRS) to represent the meaning of a dialog between the robot and a human. DRS is built on a subset of first-order logic, and besides the ability to model dialog and solve references, logic inference can be used to detect inconsistencies and rule out interpretations. The dialog system consists of a collection of agents within the Open Agent Architecture. It consists of a speech recognizer, a speech synthesizer, a dialog manager, a dialog history among others.

The system has been tested in a real environment with satisfaction.

### 4.3.5 Carl

Carl [41] is a robot and a project that aims at integrating communication, action, reasoning, and learning into an animate, adaptable, and accessible intelligent robot. By *animate* one means that the robot should respond to changing conditions in the environment. It should be *adaptable* in the sense that it adapt to different users and different physical environments. This includes reasoning and decision-making at the task-level, and learning capabilities. By *accessible* one means that it should be able to explain its beliefs, motivations, and intentions, and it should be easy to command and instruct.

Carl is an 85 cm tall robot with two drive wheels and a caster. It includes wheel encoders, front and rear bumpers rings, front and rear sonar rings, IR sensors, an audio I/O card, and a pan-tilt camera. It also carries a microphone and a speaker.

Carl has a simple dialog system including a speech recognizer, a speech synthesis, and a semantic parser. The dialog system is used to teach Carl new information. The user could for example tell Carl that **Professor Doty is in Portugal**, and Carl saves the information and can later on perform a correct answer to the question **Where is professor Doty?**. The robot also uses the dialog system to learn characteristics of new objects. It can for example learn the concept **person** by asking **Is this a person?** for every obstacle it encounters. Based on the obtained answer and visual feedback, Carl feeds the information into a backpropagation neural



network.

## 4.4 Planning in Dialog Systems

Planning in dialog systems has recently gained interest, both in traditional dialog systems and in case-based approaches. Planning can both be used as a tool to create systems that can perform more natural dialog and to create actual planning assistants which help the user with a planning task. In this section, a collaborative planning assistant called TRIPS is described. In section 4.5 other systems that integrate case-based reasoning and case-based planning are described.

### 4.4.1 TRIPS

TRIPS [11], [12], [29], The Rochester Interactive Planning System, is a collaborative planning assistant. It is designed to be a general system for assisting a human manager to construct plans in crisis situations. It is built on the agent technique where each component is seen as an agent and communicates by exchanging KQML messages [31]. The components of TRIPS can be divided into three groups:

- *Modality Processing*: This includes speech recognition and generation, graphical displays and gestures, etc. All modalities are treated uniformly, and their representations are based on treating them as communicative acts.
- *Discourse Management*: These components are responsible for managing the ongoing conversation, interpreting user communication in context, requesting and coordinating specialized reasoners, and selecting what communicative actions to perform in response.
- *Specialized Reasoners*: These components solve hard problems such as planning courses of actions, scheduling events, or simulating the execution of plans.

TRIPS performs a semantic parsing on the utterance from the user. The parse tree is sent to the *Interpretation Manager*. The *Interpretation*

*Manager* interprets the utterance and generates updates to the *Discourse Context*. The interpretation involves identifying the intended speech act, the collaborative problem solving act that it furthers, and the system's obligations arising from the interaction.

It invokes the *Reference Manager* to attempt to identify likely referents for referring expressions. The *Reference Manager* uses the accumulated discourse context from previous utterances plus knowledge of the particular situation in order to identify likely candidates.

The *Interpretation Manager* then uses the *Task Manager* to aid in interpreting the intended speech and problem solving acts. The *Task Manager* supports operations intended to assist in both the recognition of what the user is doing with respect to the task at hand and the execution of the problem solving steps intended to further progress on the task at hand.

The *Behavioral Agent* is responsible for the overall problem solving behavior of the system. It decides what the system should do according to both the interpretation of the user's utterances and actions in terms of problem solving acts, the persistent goals and obligations of the system, i.e. to furthering the problem solving task, and exogenous events of which the *Behavioral Agent* becomes aware.

The *Generation Manager*, which performs content planning, receives problem solving goals requiring generation from the *Behavioral Agent* and discourse obligations from the *Discourse Context*. The task of the *Generation Manager* is to synthesize these input resources and produce plans for the system's discourse contribution.

TRIPS has been tested in several different domains such as planning an evacuation of a population on an island, and emergency resource scenarios.

## 4.5 Case-Based Reasoning and Dialog

Case-based reasoning has been shown to be useful in several different ways regarding dialog systems. Conversational case-based reasoning (CCBR), described in section 3.3.2, is a well defined area of research. SiN is an example of a dialog system that combines CCBR techniques with planning. The Discourse Goal Stack Model integrates a discourse model with CCBR.

However, dialog and CBR does not only imply CCBR. Systems that use

the CBR architecture to implement dialog systems where the dialog is the main case problem, not only an aid to query the user for more information about the problem, has been suggested by, e.g., Murao et al.

#### 4.5.1 SiN

Most planners require full domain knowledge to solve a planning problem. However, in many planning domains, developing a complete domain theory is infeasible. SiN [9], [44], is a case-based planning algorithm (see section 3.3.1) that combines conversational case retrieval with generative planning. SiN integrates the SHOP generative planner [47] with NaCoDAE, a conversational case retriever [20]. NaCoDAE works as described in section 3.3.2. The algorithm is provably correct and does not require a complete domain theory nor complete information about the initial or intermediate world-states.

SiN receives as input a set of tasks, a state, and a knowledge base consisting of an incomplete domain theory and a collection of cases. The set of tasks is the problem to be solved. Initially the task set consists of compound tasks that have to be decomposed. For every compound task in the task set, SHOP tries to decompose it. If that can not be done, NaCoDAE takes over and tries to find cases that are able to decompose it. Suiting cases whose preconditions are satisfied, are listed, ordered by some best-first priority. If several cases are listed, the conversational part of the system kicks in. The system may need more information to be able to distinguish between the cases. This information can be obtained from the user by asking relevant questions. The user may select one of the displayed questions and answer it. The answer is then used by NaCoDAE to remove and reorder the cases. If NaCoDAE does not have any cases to decompose the task or if the user decides not to apply any applicable case, NaCoDAE will cede control to SHOP. If neither SHOP nor NaCoDAE can decompose the task, SiN will backtrack, if possible. If the task is one of the initial problem tasks, backtracking is impossible, and the planning process interrupts and returns failure.

SiN has been tested in several domains, including the *Noncombatant Evacuation Operations Domain* (NEO). NEOs are military operations for evacuating non-military persons that are in danger to an appropriate safe

haven. Information about NEOs can be found in NEO doctrines, case studies, and more general analyses. The absence of a complete domain theory makes it suitable as a test domain for SiN. The experiments showed that SiN is capable of allowing the users to guide the planning process towards their preferences while dynamically capturing world-state conditions.

### 4.5.2 The Discourse Goal Stack Model

The Discourse Goal Stack Model (DGSM) [19] is a discourse model integrated in conversational case-based reasoning. It can handle temporal topic shifts and subdialog. In a CCBR system, subdialog can occur when the system asks the user questions to be able to select the most suiting case, or when the user or the system asks clarifying questions.

The model uses text in and text out. It is based on the view of CCBR as a specialized form of goal-oriented dialog. The goals can, e.g., be to select appropriate cases, ask questions, etc. DGSM consists of the following constructions:

- A *goal stack* that permits all dialog goals to be handled in a uniform fashion and handles interruptions and subgoals.
- A *collection of discourse goal types*, i.e. what kind of goal types that are allowed on the goal stack.
- A *forest of augmented transition networks (ATNs)* in which nodes are discourse goals and arcs are speech acts by the user or the system. Every question type in the CCBR system, and every clarifying question type in the system has a corresponding ATN, which describes the question and answer types allowed to perform a legal dialog.
- A *goal handler* that is responsible for determining the appropriate action to take in response to the goal at the top of the stack.

The goal handler is the main loop of DGSM. Depending on the user utterance and the top of the goal stack, it decides what action to perform. The action could for example be to generate a speech act, or to invoke the CCBR module. The DGSM CCBR module finds the most discriminating question amongst the appropriate cases, finds the ATN for that particular

question and pushes the first node on the goal stack. The goal handler generates the speech act and pushes the expectation of an answer to the question on the goal stack. If the user instead of answering the question asks a clarifying counter question, the goal handler finds the corresponding ATN and pushes it on top of the stack. The expectation of an answer to the original question from the system is now waiting further down in the stack and the answer generation is in focus. When the user gets the answer to the clarifying question she may be able to answer the original question that is now on top of the stack again. When the original question is answered, the CCBP module can use the answer to select a unique case or to further minimize the case set by asking other discriminating questions.

DGSM is implemented in RealDialog, a web-based conversational agent for customer relationship management.

### 4.5.3 Case-Based Dialog Systems

Case-Based Reasoning has been used not only in CCBP systems but as a base architecture for dialog systems. Murao et al [45], [46] presents a case-based dialog system for information retrieval. Dialog examples are retrieved from human-to-human dialog and from wizard-of-Oz experiments. In a wizard-of-Oz experiment, the user thinks she is speaking to an intelligent machine but in reality there is a hidden person behind the answers from the machine. Such experiments show how a user would interact with an intelligent machine although no such machine has been constructed yet. The retrieved dialog corpus is used to construct the system's case base.

The system consists of the following modules:

*The Dialog Example Database* is the case base in the CBR system. The cases consist of a problem part and a solution part. For system input the problem part is a user utterance and the solution part is a query which searches a database for the information wanted. For system output the problem is a database answer to a query and the solution is an answer in natural language. A system input case is grouped together with its system output case to model the whole dialog.

*The Word Class Database* consists of the important words classified semantically. The classifications are based on the dialog corpus.

*The Target Information Database* consists of the target information the user is querying.

*The Speech Recognition Module* creates a transcription from the user's verbal input.

*The Query Generation Module* extracts the case that is the most similar to the present input from the Dialog Example Database. Then the case query is modified to suit the present input.

*The Search Module* executes the query in the target information database.

*The Reply Generation Module* extracts the case that is the most similar to the present search result from the Dialog Example Database. Then the case reply statement is modified to suit the present situation.

*The Speech Synthesizer* synthesizes the sound of the reply statement.

Words that characterize the meaning of the utterance are used for the similarity calculation. Important words are classified using the information in the Word Class Database. For each problem formulation in an input case, and the present input, the number of matched non-classified words and the number of important words belonging to the same word class is accumulated with the corresponding weight, and the result is treated as the similarity. The case which marks the highest similarity is selected.

The query for the most similar case is retrieved and adapted according to the present input. The modification is performed by replacing the keywords in the query with words in the input utterance if they belong to the same word class.

The similarity and adaptation of the query result is produced in a similar manner.

The system has been tested in a domain where the user is driving a car and queries the system for restaurant information. Interesting attributes are, e.g., what type of food is served, how popular the restaurant is and how close it is to the user. The system provided the correct query in about 64% of the cases and the partially correct query in about 88% in these tests.

Other case-based dialog systems presented in the literature are, e.g., a system for learning to sustain a natural conversation without constraining the topic [34], and a dialog system that uses not only the utterance but also facial expressions to express past cases [57].





# Chapter 5

## Phase I - The WITAS RDE

### 5.1 Background

As mentioned in section 4.1, several dialog systems have been developed within the WITAS project. The work with the WITAS-Stanford dialog system presented in section 4.1.2 ended in year 2002, and the Cognitive Autonomous Systems Laboratory at Linköping University started to develop a new dialog system called The WITAS Robotic Dialog Environment (RDE) inspired by the WITAS-Stanford Dialog System. The development of the WITAS RDE is a joint effort between several members of the laboratory, and the author of this thesis has taken a major part in the development of the dialog manager and the test simulator.

### 5.2 The WITAS RDE Architecture

The WITAS RDE is a big system consisting of a number of loosely coupled subsystems:

- The *Autonomous Operator's Assistant*, AOA, consisting of two parts:

a *Speech and Graphical User Interface*, SGUI, and the DOSAR dialog manager.

- A *Robotic Agent* consisting of the actual physical robot and data from its sensor and onboard systems or a simulated robot in a simulated environment [13].
- A *Development Infrastructure* that provides the services that are needed for the development, demonstration, and validation of the dialog system.

### 5.3 The Autonomous Operator's Assistant

The AOA is inspired by the WITAS-Stanford Dialog System. It marks the second phase of the dialog project connected to the WITAS project and is seen as the next generation dialog system. It differs from the WITAS-Stanford system mainly due to its advanced multimodal user interface and logic base. The AOA uses Cognitive Robotics Logic (CRL) [54] to obtain a representation of action and time. CRL is developed by Erik Sandewall and has its origin in the book *Features and Fluents* [5]. Within CRL, time is a central notion, which is reflected in the dialog system.

Several kinds of actions are represented in a uniform manner, namely:

- Physical actions by the helicopter as a whole (take off, land, fly to X etc) and by its various components such as its video camera system.
- Speech acts and other communication acts, as performed by both the operator and the dialog system
- Cognitive acts, such as parsing a sentence, or deciding what to say next.

There are several reasons for representing these types of actions in a uniform way, e.g., it reduces the number of concepts that have to be considered in the system. Another reason is to prepare for dialog where the user makes combined reference to several types of actions, for example **where were you when I told you to fly to the parking garage?**. A third

reason is the ability to concisely represent the succeed/fail distinction in the cognitive actions that are involved in the dialog.

A natural way to represent the most common actions in a dialog system is by the following classification:

- *The speech act proper:* The period of time where the phrase is actually uttered.
- *The understanding of the phrase:* The phase where the phrase is converted from speech to text by the speech recognizer, the parsing, identification and screening of semantic content, etc.
- *The decision how to react:* The phase where the system decides what would be the most appropriate reaction. It could for example be to answer a question, to send a command to the helicopter, or to verify that a command has been correctly understood before it is sent to the helicopter.

However, despite the similarities between the action types, they are different in two important aspects: the time scale and the effects of the actions. The time scale may vary considerably between the actions, from several seconds for a speech act proper down to a couple of milliseconds for the decision how to react. The effect of an action depends on the character of the action. A physical action has effects on the state of the world. Cognitive actions, on the other hand, can better be thought of as transformers that convert one symbolic expression (for example, a parse tree) to another symbolic expression. These kinds of similarities in actions are not well dealt with in the traditional formalism of action and change, but the CRL formalism has been extended to cope with such differences.

### 5.3.1 The SGUI

The front end of the WITAS RDE is the SGUI application. It is a user interface which provides both facilities for using speech or text. It consists of an image of the area where the helicopter is flying, serving as a map for the operator, and a facility to show a video stream from the camera onboard the helicopter. Figure 5.1 shows SGUI when it is playing a video.



Figure 5.1: The SGUI showing a video.

The video can be live or an earlier recording saved in the video server. It is multimodal, which gives the user the possibility of using gestures on the screen such as points and figures for deictic reference. The gestures can be performed both on the still image and on a video stream. The domain most commonly used throughout the WITAS project is the traffic surveillance scenario, and the following four types of gestures are implemented in the system:

- Indicate a particular point in the image, for example for a fly-command.
- Indicate a particular area in the image, for example for a command to survey the area or to not fly over it.
- Indicate a particular trajectory in the image, for example a segment of a road that the UAV is to fly along, or patrol back and forth.
- Indicate a particular vehicle or other moving object that is part of a query or command to the UAV, for example that the UAV should follow it.

The gesture part of the SGUI interprets the movements of the pen or the mouse, and attempts to classify the input according to these four cases. Figure 5.2 shows an example of SGUI where the operator has indicated a point in the map, at the same time as saying the command **Fly here**. The point is indicated with a cross in the map. If the gesture was made on a video stream, the coordinates of a referenced point must be identified and transformed to a point in the actual world. If it was a moving object that was referenced, the position of the object has to be identified. In the case of live video, this has to be done by a video interpretation in the UAV itself. However, the image recognition system onboard the helicopter is not yet able to do such a transformation, hence the test videos are manually tagged. When testing against a simulated robot in a simulated world, these parameters can be generated as a by-product of the visualizer.

If the user used speech as a means of communication, SGUI sends the sound file to a speech recognition system, in this case the Nuance<sup>1</sup> system. Nuance returns one or several conceivable interpretations of the utterance together with a confidence score for each of them. The confidence score indicates how confident the speech recognition system is for each of the interpretations. If the user provided the system with gesture data, it is interpreted as one of the four types of gestures mentioned above and combined with the suitable language input and sent as a request to the DOSAR dialog manager for evaluation. Some gestures, such as a gesture showing three-quarters of a circle, may either designate an area or a trajectory, and

---

<sup>1</sup><http://www.nuance.com/>



Figure 5.2: The SGUI after the operator has pointed on the map.

the correct choice can only be interpreted in combination with the accompanying phrase. In those cases, SGUI first inspects the phrase in text form and tries to match it against some common predefined sentences. If the phrase matches, the corresponding gesture interpretation is provided in the request. If the SGUI can not solve the interpretation problem directly, it makes an assumption and provides it together with the phrase as a request to DOSAR. If the latter should decide that the SGUI's interpretation of the

combined speech and gesture was incorrect then it sends a message back to the SGUI asking it for a new interpretation of the gesture based on the alternative classification.

SGUI receives output messages from DOSAR, either as a response to a request or as a response on a message sent from the robotic agent to DOSAR. The message is in the form of a text message in natural language, and it is directly sent to a speech generation system which generates and plays a corresponding sound file. Two speech generation systems have been used, namely Festival<sup>2</sup> and BrightSpeech<sup>3</sup>.

### 5.3.2 The DOSAR Dialog Manager

DOSAR is the dialog manager in RDE. It receives messages from SGUI or the robotic agent written in a KQML like form [31]. DOSAR consists of the following units:

- A set of *consumers* that handle the message sending from various parts of the DOSAR system and also from and to SGUI and the robot/simulator.
- A *parser* that transforms the text input into a parse tree. The grammar used in the DOSAR parser is the same as the one used by the speech recognition system.
- A *salience list* consisting of the referenced objects ordered by priority.
- A *semantic analysis unit* that checks the semantics in the parsed sentence. It uses the salience list to resolve references in the input.
- A *speech translation unit* that decides how to translate the input to a request that can be sent to the robot. If the input is a question that DOSAR can answer directly, an answer in an intermediate form is constructed. The speech translation unit uses the parse tree and the semantic information produced in an earlier stage, to find a suitable set of functions to execute. The functions create the request to the helicopter or a response message in an intermediate form.

---

<sup>2</sup><http://www.cstr.ed.ac.uk/projects/festival/>

<sup>3</sup><http://www.brightspeech.com/>

- A *robot translation unit* that decides how to translate the response or output from the robotic control system. A label on the message tells DOSAR what kind of output it is. It can for example be a response to a command or an observation. Depending on the label, DOSAR decides if the information should be forwarded to the operator. In that case, an output message in an intermediate form is created.
- An *output buffer* where the messages to be sent to the SGUI are temporarily stored. The messages in the output buffer can be inspected, several messages can be merged into one message to achieve a more natural dialog, and messages can be prioritized. When the most suitable message to send to the SGUI is chosen, it is transformed from its intermediate form to an actual output sentence in natural language.
- A *memory* where each input and output are stored and connected to each other corresponding to the actual dialog taken place. Observations from the helicopter that are not reported to the operator are also stored in the memory.

The main dialog message flow in DOSAR is shown in Figure 5.3. It does not show the salience list and the memory because they are only containers.

Syntactic parsing, as described in section 2.3.1, is used as the syntactic model. It produces a parse tree which is used in the semantic analysis unit. The dialog model used is a simple and implicit form of dialog grammars, see section 2.4.1, where a request from the operator is identified using the information in the parse tree. When the request is identified, a matching response is created by the system. If the semantic meaning of the request can not be uniquely identified, a clarifying question is created and a structure to catch the answer to the question, including saving discourse information about the preceding dialog, is triggered. When an operator input arrives that may be the answer to the question, the structure catches it and uses the saved discourse information together with the new information to resolve the original ambiguous request. An example of several levels of subdialog is shown in Figure 5.4. It is also an example of the **abort** command which immediately discards all actions causing the helicopter to start hovering.



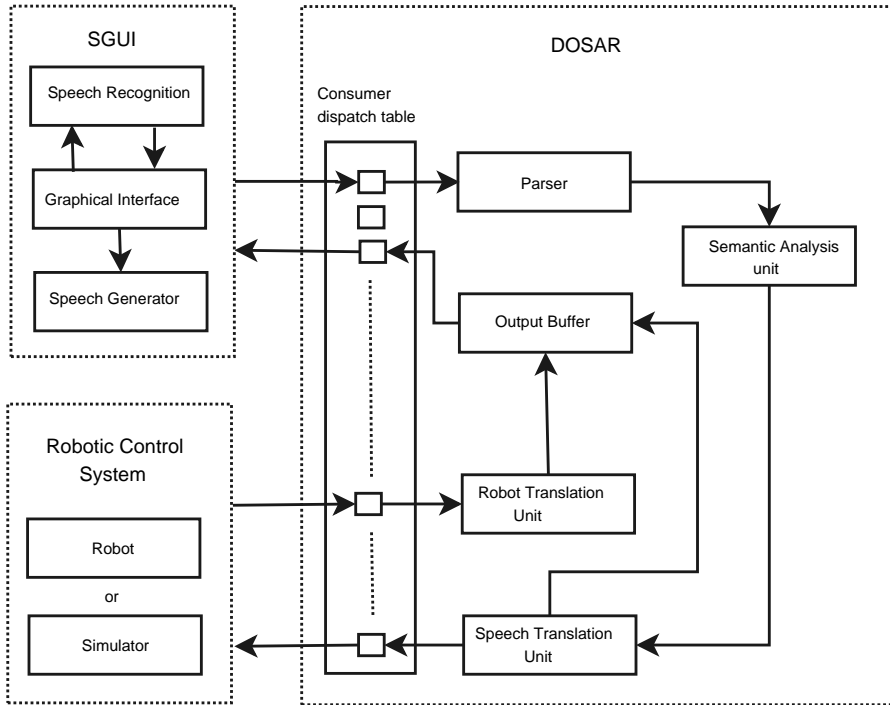


Figure 5.3: The main dialog message flow in DOSAR.

The system also handles sequences of commands, grouped together with the word **and**, as is also shown in Figure 5.4.

References to objects mentioned earlier in the dialog are resolved using a salience list. Whenever the system finds an object that may be referred to later in the dialog, information about it is saved on the salience list. When a reference is detected, it is assumed that it refers to the latest mentioned object, which is found on top of the salience list. The dialogs in Figure 5.5 and in Figure 5.6 show examples of this.

---

Operator: Fly to the school.  
DOSAR: The helicopter has 2 schools to choose between.  
DOSAR: Which school do you mean?  
Operator: Which schools can I choose from?  
DOSAR: You can choose between the green wood school and  
the red brick school.  
Operator: The red brick one.  
DOSAR: The helicopter is about to start flying to the red brick  
school.  
Operator: Abort.  
DOSAR: The helicopter has aborted.  
Operator: Fly home and land.  
DOSAR: The helicopter is about to start flying home.  
DOSAR: The helicopter has completed flying home.  
DOSAR: The helicopter is about to start landing.  
DOSAR: The helicopter has completed landing.

---

Figure 5.4: An example dialog demonstrating subdialog and sequences of commands using `and`.

The memory plays several different roles in the DOSAR system. It is used as a history of what has happened during the current and past system executions and when it happened. Observations that are reported from the UAV are stored in the memory and can be queried by the operator, e.g., `when did the bus overtake the blue volvo car`. The information can also be used as a form of information tagging of the recorded video stream, as demonstrated in the dialog in Figure 5.5. The timestamp of an event can be obtained from the memory and the video segment showing that actual time slot can be requested from the video server and shown by SGUI. The memory also contains discourse information about the dialog that has been performed, see section 2.5 about dialog histories. Requests and their corresponding responses are connected using the information from the implicit dialog grammar. The request-response pairs are further connected on a higher level to represent the overall dialog belonging to the

---

Operator: Fly to the hospital.  
DOSAR: The helicopter is about to start flying to the hospital.  
Operator: What is the velocity?  
DOSAR: The velocity is 3 meters per second.  
Operator: Is the helicopter there yet?  
DOSAR: The helicopter is 176 meters from the hospital.  
Operator: Rotate the camera forty degrees left.  
DOSAR: The helicopter is rotating the camera.  
Operator: Does the helicopter see any buildings?  
DOSAR: No, the helicopter does not see any buildings.  
DOSAR: The helicopter has completed flying to the hospital.  
Operator: Ascend fifty meters.  
DOSAR: The helicopter is about to start ascending 50 meters.  
Operator: What is the altitude?  
DOSAR: The altitude is 41 meters.  
Operator: Show me the video from when the helicopter flew to  
the hospital.  
DOSAR: Here it comes.

*The video appears in SGUI*

---

Figure 5.5: An example of referencing using the word **there** and video playing using the memory.

same topic. This dialog history is used in the output buffer to prioritize the output messages waiting to be sent to speech synthesis. The messages are prioritized to stick to the current ongoing dialog about a topic, before changing to another topic, that is, new outgoing messages that are connected to the last utterance in the memory, have precedence over messages belonging to other dialogs. If, for example, the operator asks the system a question while the system creates a status message that tells the operator that the helicopter has completed a fly manoeuvre, the answer to the question is given higher priority hence it is formulated and sent to the speech synthesis before the status message.

DOSAR contains a world representation of its own, which does not have to be fully accurate with respect to the actual world, nor to the world representation in the robotic agent. The representation is an ontology where static objects and their attributes such as color or positions are stored. Examples of objects in the traffic surveillance domain are buildings and roads. The information in the ontology corresponds to the world knowledge known by the dialog system at a particular time. When a request includes a reference to, e.g., a hospital, the dialog manager searches the world representation for a matching item and returns the internal identifier for the building. An internal identifier is a unique string which serves as the name of the object. The agent has the same unique identifiers for the information that it has stored about the object, hence the dialog system can send a fly-to command where the destination of the fly-command is given as an internal identifier. The dialog system may also compute a position near the referred object and send it as the destination if that is preferable. If the system finds several matching items when looking up the internal identifier, it has to start a clarifying subdialog with the operator.

The operator can add new instances of objects to the world representation in DOSAR by explaining what kind of object it is and where it is located in the world, e.g., **there is a container south of the hospital**. DOSAR adds it temporarily to the world representation and when it gets a visual confirmation of the object from the camera system of the agent, it is permanently stored in the ontology. Figure 5.6 shows an example of a dialog where this is used.

The DOSAR system is provided with a small simulation module which can perform a fast simulation of a scenario. The information obtained is used as basic data for answers to estimation related questions such as **can the helicopter fly to X with 0.5 liters of fuel under 3 minutes**. The simulation module is very limited and simple at the time of writing, and one idea is to integrate the Hazard Simulator described in section 5.4, and increase the functionality to also include fast simulations of an event. The fast simulations can be used for reasoning about actions and time and for planning different actions under certain constraints. This is however not realized in the dialog.

Apart from understanding dialog where the operator gives commands and the robot performs them, DOSAR also manages simple planning prob-

---

Operator: There is a body south of the church.

Operator: Fly to it.

*The helicopter saves the information about the body temporarily in the world representation, until the information is confirmed.*

DOSAR: The helicopter is about to start flying to a body.

Operator: How far is the helicopter from the church?

DOSAR: The helicopter is 141 meters from the church

DOSAR: The helicopter has completed flying to a body.

Operator: Does the helicopter see any body?

DOSAR: Yes, the helicopter sees a body.

*The information is stored permanently in the world representation.*

---

Figure 5.6: An example a dialog where the operator presents new information to the system.

lems. An example is given in Figure 5.7. In this particular example, the helicopter needs an ice sensor and a camera, but it can only carry one device at a time, hence the planner has to plan the best way to perform the tasks with respect to getting and dropping devices at the home base.

## 5.4 Simulations and Test Flights

The ultimate test of the AOA is of course to connect it to the actual UAV for flight control. This is however not always practical due to several reasons.

- It is a safety risk to perform initial tests on the actual helicopter, in case the system is malfunctioning.
- It is time consuming and impractical to perform actual flight tests because the UAV is only allowed to fly in restricted areas, where the UAV and equipment have to be transported.

---

Operator: We shall plan mission two.  
DOSAR: Ok, what are the problems and the tasks at this point?  
Operator: Check ice on road along Main street.  
DOSAR: Ok.  
Operator: Check a reported accident at the church.  
DOSAR: Ok.  
Operator: Schedule the mission.  
DOSAR: Ok, I have determined a schedule for this mission.  
Operator: Start mission two.  
DOSAR: Ok.  
DOSAR: The helicopter has arrived to the church.  
DOSAR: The helicopter has completed obtaining pictures of the accident.  
DOSAR: The helicopter has completed flying home.  
DOSAR: The helicopter has completed unmounting present payload and mounting the new payload.  
DOSAR: The helicopter has completed flying to the Main street.  
DOSAR: The helicopter has completed turning on the ice sensor.  
DOSAR: The helicopter has completed flying along the Main street.  
DOSAR: The helicopter has completed turning off the ice sensor.  
DOSAR: Mission two has now been completed.

---

Figure 5.7: A planning dialog.

- UAV architecture experts and a human pilot that can control the UAV remotely if necessary, need to participate in the tests.

Some complex dialogs include information about objects and the world environment that the UAV sensors and systems are not yet able to perform. These constructions have to be exclusively tested in a laboratory environment until the onboard system can provide the dialog system with the information needed.

Within the WITAS RDE project, a simulator called Hazard [14] has been developed. The user interface of the simulation is shown in Figure 5.8. The Hazard simulator includes a simulated UAV situated in a simulated



Figure 5.8: The Hazard simulator.

environment. The simulated UAV can perform high level actions such as flying to a building with a certain identity, flying in a certain direction, tracking a specific car, flying along a specific road, take off, land, ascend and descend, etc. It reports the results of an action and also reports if it observes any buildings as it flies. It is able to answer status questions such as the current altitude, velocity and heading. The simulation of UAV movements and environment goes only to the level of detail that is required for the dialog, and does not go down to the level of exact flight dynamics. The purpose of Hazard is to be able to test both those dialogs that can be performed using the actual UAV, and those dialogs that for example

require complex image processing.

Dialog that can be performed by the UAV can be further tested in the simulation of the robotic control system used to test the architecture described in 4.1. This is a lower level of simulation than the one offered by the Hazard simulator. Both the Hazard simulator and the WITAS simulator provide an interface which is very similar to the one offered by the actual WITAS UAV, in order to make it straightforward to use the AOA during actual flights.

An early version of the AOA was tested during actual flights in late 2003, when the main WITAS DEMO was performed in the presence of an international evaluation committee. The AOA dialog system, including the DOSAR dialog manager and the SGUI user interface, was successfully connected to the WITAS UAV architecture, and high-level voice commands using natural language have been used to control the onboard camera. Further development has been performed since then, in particular with respect to the SGUI.



## Chapter 6

# The Research Problem for phase II

### 6.1 Ideas Behind the Research Problem

Natural language is the preferred means of communication between people, and well functioning natural language dialog systems relieve the user from the burden of learning an artificial language to communicate with computers. The keywords are however *well functioning*, and the dialog system area has a long way to go before we can create dialog systems that perform dialog as well as a person. What is needed to improve the dialog systems?

The dialog manager needs a discourse model, as discussed in section 2.5, to be able to resolve references, manage subdialogs, manage dialog topics and goals, and to manage topic changes. Without a discourse model, the dialogs tend to be simple, limited, and strongly structured by the system.

Besides a discourse model, the system needs domain knowledge and dialog knowledge, as well as methods to make use of the knowledge intelligently. The knowledge can be provided to the system explicitly, or obtained implicitly by the system, using past experience. In the former case, the knowledge is often written by hand, which is tedious and time consuming. There is also a risk that some useful information can be omit-

ted by mistake, or that the type of information needed will change from the time of writing. In the latter case, the system needs a method to acquire information automatically and to reuse it.

The work with the WITAS RDE in phase I presented in chapter 5, and in particular with the DOSAR dialog manager presented in section 5.3.2, gave inspiration and pointed out some interesting problem areas. Every new command or dialog needed to be specified by hand in the system, even if it was similar to other dialog constructions. It would be better if the system could figure out by itself how to react on new commands, given the already known information. The system should also make effective use of the knowledge, and combine it if necessary, to solve new problems in a flexible manner. With such flexibility, the system could adapt gradually to an ever changing environment and to changing tasks. The memory in the DOSAR system was an ad hoc solution with potential. Could it be used as a knowledge base over the dialog history? That is, both as a discourse model and as a case base? The idea was promising and definitely worth more investigations.

## 6.2 Problem Formulation

The problem we address can be captured concisely in the following question:

*How can an adaptable, learning, problem solving robotic dialog system be constructed so that it improves over time and incorporates advanced natural language features?*

This question can be decomposed into the following subquestions:

- How can **machine learning** be provided in such a system?
- How can **problem solving** be provided in such a system?
- How should the **discourse information** be represented?
- How can the system make **use of the knowledge to solve problems it has never seen before?**

- How can the system **learn from the user** if it can not solve a problem on its own?
- How can **special considerations regarding the physical robot**, such as safety and graceful degradation, be taken care of?

A continuation of these questions is to ask which methods are best for each of these items, assuming at least one method is available and suitable. This question is not answered in this thesis, but serves as an inspiration for further work.

In the following chapter, a system whose design addresses these issues is presented.



# Chapter 7

## Phase II - CEDERIC

### 7.1 Introduction to CEDERIC

CEDERIC is an abbreviation for Case-base Enabled Dialog Extension for Robotic Interaction Control. It is a dialog system designed for dialog with a physical robot, in particular the WITAS autonomous unmanned aerial vehicle that was described in section 4.1. To recapitulate, the WITAS project focuses on the development of an airborne computer system that is able to make rational decisions about the continued operation of the aircraft, based on various sources of knowledge including pre-stored geographical knowledge, knowledge obtained from vision sensors, and knowledge communicated to it by data link.

CEDERIC is a dialog manager based on the experiences gained from the work with the WITAS RDE described in chapter 5. It can be used in place of the DOSAR dialog manager and uses the SGUI user interface and the Hazard simulator. It can also be used in combination with DOSAR, where CEDERIC takes over if DOSAR does not find any proper reaction on an input.

CEDERIC uses CBR, described in section 3.3, both as a problem solving algorithm and as a machine learning algorithm. The case base consists of two different sorts of cases:

- *Plan cases*, where the problem part is the user utterance and the solution part is a plan consisting of a sequence of plan items.
- *Plan items*, where the problem part can be a subset of the user utterance or computed information that is the result from other previously executed cases. The solution can be, for example, a computation of a value, a robotic control command, or a reply in natural language that is to be sent back to the operator.

An utterance from the operator is matched against the problem part of the plan cases, and the plan of the most similar case is executed if the case is similar enough. If it is not similar enough, several different cases can be combined using CBP, described in section 3.3.1, to solve the problem. The most similar case is used as a basic case and plans from other cases are combined with it to solve the problem. If such a combination is found, it is executed, and the new problem and the corresponding solution are saved in the case base for further use. The case base will grow over time and be able to solve more problems, using its new information and experience. This method allows CEDERIC to use the information in the case base efficiently, combining different cases to solve a new problem.

CEDERIC consists of a *lexicon*, a *case base*, *domain knowledge*, a *discourse module* and a *case-base manager* as shown in Figure 7.1.

The lexicon is used to classify the words in the user utterance according to some predetermined word classes. The classified utterance is matched against cases in the case base by the case-base manager. The discourse module is responsible for maintaining a discourse model of the dialog so far to be able to interpret the operator's sentences in the right context. The discourse model helps the system to interpret references which may refer to sentences earlier in the dialog and to keep track of different ongoing dialogs. The domain knowledge contains an ontology of the world as the robot knows it and a categorization of the world items. The purpose is twofold. It serves as a world representation which gives CEDERIC knowledge about which buildings there are in the known world, what kind of buildings they are, where they are located, and their attributes such as color and material. It also gives CEDERIC fundamental knowledge about categorization, e.g., which items can be called buildings in the dialog and which can not.

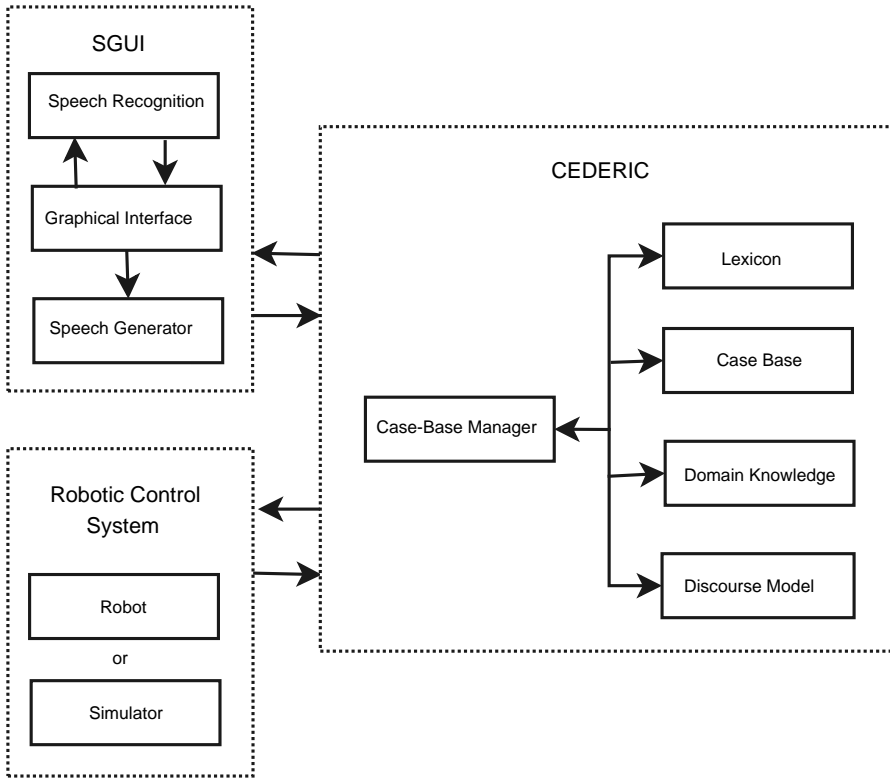


Figure 7.1: Architecture of CEDERIC.

The user interface used is the SGUI, which was described in section 5.3.1. In SGUI the operator can choose to use either speech or text for the input to the dialog system. The speech recognizer used is the off-the-shelf product Nuance<sup>1</sup> and the speech generator used is one of the off-the-shelf products

---

<sup>1</sup><http://www.nuance.com/>

Festival<sup>2</sup> or BrightSpeech<sup>3</sup>. When learning a new word using speech recognition, one can choose between having a considerably bigger grammar for the speech recognition than the dialog manager and only consider learning in the dialog manager, or provide the new word in text form in the learning phase and then compile it into the speech recognition grammar if that can be done at runtime. The system supports both approaches.

Regardless of whether a speech recognizer is used or not, a sentence from the operator arrives to CEDERIC in plain text format. It is classified, processed in the CBR engine and CEDERIC returns either a new phrase in text format to be sent to the speech generator, or a request to the robotic control system. The interface between the robotic control system and CEDERIC is identical to the one used by the WITAS RDE, described in chapter 5. The responses that the robot sends back to CEDERIC are sent on a different channel than the one the operator uses. This is because the response should not be classified in any way, but be processed directly in the CBR engine.

## 7.2 Design Choices

Several design choices have been taken during the development of CEDERIC. The use of CBR and CBP is vital for the system's functionality and has been guiding design choices in other areas as well. In this section, the choice of CBR and CBP is motivated, and some comparisons are made between the the dialog system theory described in chapter 2 and design choices made in CEDERIC regarding these areas.

### 7.2.1 The CBR Architecture

As already mentioned, CBR is used as the main architecture of the system. It works mainly as a problem solving method to find a suitable response to an input. However, the capabilities to solve new problems increases as new cases are stored in the case base as a result of the system's experiences, and therefore CBR is also used as a machine learning algorithm.

---

<sup>2</sup><http://www.cstr.ed.ac.uk/projects/festival/>

<sup>3</sup><http://www.brightspeech.com/>



CEDERIC and the WITAS AOA, described in section 5.3, have the same domain and hence their various types of actions are similar. The list of actions in CEDERIC is an extension of the AOA action list:

- Physical actions by the helicopter as a whole (take off, land, fly to X etc) and by its various components such as its video camera system.
- Speech acts, performed by both the operator and the dialog system.
- Cognitive acts, such as fetching information from the domain knowledge or creating action commands.
- State information retrieval from the helicopter, such as altitude, direction, and velocity.

As in the AOA system, the actions in CEDERIC are represented and handled in a uniform manner. However, unlike the AOA system, CEDERIC uses CBR, where the actions are generally described and handled as cases in the case base. This provides the dialog system with a simple and modular design. New functionality is directly added by writing new cases and storing them in the case base. New domain knowledge similar to existing knowledge can be added to the system in a simple manner. It can directly be used by the system without any additional changes to the case base or the case-base manager, due to the flexible and adaptable nature of the CBR design. This provides us with the facility of letting the system incorporate new information into the system automatically, such as new words or knowledge about the physical world. This knowledge can then be used directly by the cases in the case base, giving the system mechanisms for updating its own knowledge and increasing its performance. The new information can be obtained from dialog with an operator.

### 7.2.2 Case-Based Planning

The CBR architecture is also suitable for implementing CBP, described in section 3.3.1. In CEDERIC, CBP is used when no matching case is found in the case base. The most similar one is used as a starting point for the planner, which searches the case base for additional cases that can be combined with the basic case and hence solve the problem. If such a

combination is found, it is executed, and the new problem and the corresponding solution is saved in the case base for further use. This planning technique is very useful in CEDERIC for solving new, unseen composite problems by combining several different plans into a composite one.

CBP is also useful for CEDERIC to understand implicit information and to extend a plan when appropriate. One example is if the operator tells CEDERIC to fly somewhere but the UAV has not taken off yet. Then CEDERIC can understand the implicit command to take off and then perform the fly command. CBP is also a good choice when it comes to describing the plan and how the system came up with it, because the problem solving strategy is easy to describe in natural language. To make the planning more reliable, it is easy for the dialog system to report the plan and how it was constructed to the operator, and ask for confirmation before executing it. This is however not yet implemented in CEDERIC.

It has been argued that the CBP technique, where perviously stored plans are adapted to a new problem, does not add anything considering time complexity compared to generating new plans from scratch [48]. In the robotic dialog domain however, low time complexity is not the most important reason for reusing old plans. When dealing with real world problems, several factors can affect the result and all factors may not be included in the problem formulation. Therefore, it is safer to use an already proven plan as often as possible rather than generating a new one.

### 7.2.3 Syntactic Model

Syntactic parsing has been tested in CEDERIC, where the resulting parse tree was compared to the problem formulation in the cases. However, it turned out that the parsing method was too restricted to be able to use the whole potential of the CBR architecture, and the similarity measurements were hard to evaluate. Instead of using the complete parsing method, the words in a new utterance from the operator are classified using a lexicon, similar to the syntactic parsing method, but no rule set is applied to the categorized words, hence no parse tree is produced. The categories in the lexicon are described as an ontology where semantically similar words are connected to each other using the notation of classes and inheritance from the object-oriented approach. The categorizations are matched against the

categorized words in the problem formulation using a pattern matching method.

This design has several advantages. It does not require a predefined order of the words, nor totally complete input utterances as syntactic parsing does. The interpretation of the utterances is left to the CBR engine to deal with, which makes the system flexible because it can use its fully potential as a problem solving and machine learning technique. It is also rather easy to add new words to the lexicon. The drawbacks are the same as for the pattern matching method, i.e. the method can not distinguish between similar sentences with different meanings if the distinguishing word is not a keyword, and it does not create detailed information about the composition of the sentence to be used in the further steps of the dialog manager

### 7.2.4 Dialog Model

The plan-based model of dialog, where the different utterances are seen as dialog acts to accomplish some goal, was a natural choice to model dialog because it fits nicely with the CBR and CBP approaches. The advantage that the dialog can be treated as a special case of action planning is very important in a robotic dialog system, where different kinds of actions depend on each other. The purpose of the dialog is often to communicate information used to perform sequences of physical actions. The plans in CEDERIC are not pure dialog models, but also contain other kinds of actions.

CEDERIC also performs some learning of dialog policies, e.g., when new plans are formulated using CBP. This form of learning does not however build on the same principles as the Markov method described in section 2.4.3.

### 7.2.5 Dialog History

The design of the dialog history is built on the discourse model for the SmartKom project, which was described in section 4.2. The dialog history complements the dialog model with pure dialog information of the past dialogs. The SmartKom discourse model includes information used to resolve references and to keep track of several ongoing dialogs, and is also designed

to manage multimodality. Apart from that, it is easy to understand, develop and maintain the model, which makes it a good choice to use as a basic dialog history system.

### 7.2.6 Domain Model

The domain model as well as the lexicon is written as an ontology using the Knowledge Machine representation language [22]. Besides the lexicon, the domain knowledge consists of information about the world in which the helicopter is situated. The information can be topological such as positions of buildings and streets, and informational such as names of streets and types of buildings. The use of ontologies is well suited because it provides the CBR system with similarity information based on how close two entities are to each other in the ontology. The use of the Knowledge Machine language provides the system with a well defined inference and manipulation system.

## 7.3 Discourse Model

The system can manage simple cases of dialog such as a command from the user that directly produces an answer even without a discourse model, but a discourse model is necessary in order to handle a more natural and sophisticated dialog containing references to earlier objects and clarifying questions (where?, what?, which?, why?).

The following dialog problems are handled in the discourse model:

*Anaphoric references.* The discourse model should be able to solve references to objects that have occurred in an earlier stage of the dialog.

*Subdialogs.* It should be able to recognize if an utterance is a subdialog to the present dialog and hence should be interpreted within the limits of the current discourse, or if it is the start of a new dialog. It should also recognize a dialog as completed so that the old discourse is no longer applicable. It should make it possible to return to older non-completed dialogs that are not in focus at present.

*Topic management.* The discourse model should be able to figure out if the present is a good moment to mention an observed object or event, or if that utterance should wait for a better occasion when it does not disturb the present dialog.

The discourse model we have chosen to implement in CEDERIC is very similar to the discourse model in SmartKom, which was described in section 4.2. However, the discourse model in SmartKom is constructed to manage multimodal input, which CEDERIC does not yet support, hence the discourse model can be stripped and simplified to suit our needs. The CEDERIC discourse model uses similar layers as SmartKom, recapitulated here:

- The Modality Layer.
- The Discourse Object Layer.
- The Domain Object Layer.

In CEDERIC, the Modality Layer is exchanged for a Linguistic Layer that only handles objects in natural language, and the Domain Object Layer is renamed the Dialog Object Layer.

Each layer has a corresponding object type with various features. These are linked to one another in a hierarchical manner and constitute the meaning of the dialog.

*The linguistic objects.* These objects are members of the Linguistic Layer and are thus furthest down in the chain of objects. They encapsulate information about the concrete realization of a referential object, e.g., contain information about how the nouns in the dialog were uttered. They could for example have been references made using the word *it* or using a noun and a determinant.

*The discourse objects.* These objects represent concepts participating in the communication that can be referred to by other referring utterances. When a concept is introduced for the first time, a discourse object that symbolizes the concept is created. A corresponding linguistic object is also created which models how the concept was referred to

in that particular utterance. Every discourse object is unique and if the same concept occurs again in the dialog, a new object in the linguistic layer is created and linked to the discourse object. A discourse object can also be composite. An enumeration of several objects can be seen as a discourse object representing the enumeration as such, and this object contains the enumerated objects as its children. This gives CEDERIC the opportunity to understand references referring to the order of the enumerations, e.g., **the first one**. The discourse objects have links to the corresponding linguistic objects.

*The dialog objects.* These objects groups together those sentences and their information that have the same direct goal. The sentence **fly to the hospital** for example, when it is executed, gives a dialog object that groups the sentences **fly to the hospital**, **ok** and **I am at the hospital now** together. Dialog objects contain information about the topic of the dialog and which discourse objects were created due to the particular utterance.

*The global focus space.* The various objects in the dialog layer that belong to the same dialog, including subdialogs, are grouped together in a top object called the global focus space. It contains information about the main topic of the dialog, if it is ok to interrupt the dialog and which dialog objects belong to it. Each global focus space also keeps track of the discourse object last mentioned, to be able to resolve references such as **it**. This is known as the *local focus stack*. The last mentioned discourse object is said to be in focus.

To keep track of the current dialog in focus, CEDERIC saves the different global focus spaces in a stack called the *global focus stack*. The global focus space on top of the stack is said to be the one in focus. If the entire plan belonging to a global focus space has been executed, the global focus space is marked as closed and removed from the stack. Several dialogs can be open and ongoing at the same time and are thus members of the stack but only one dialog can be in focus at the same time.

Figure 7.2 shows an example of what the discourse model looks like when the utterance **Fly to the school** has been executed.

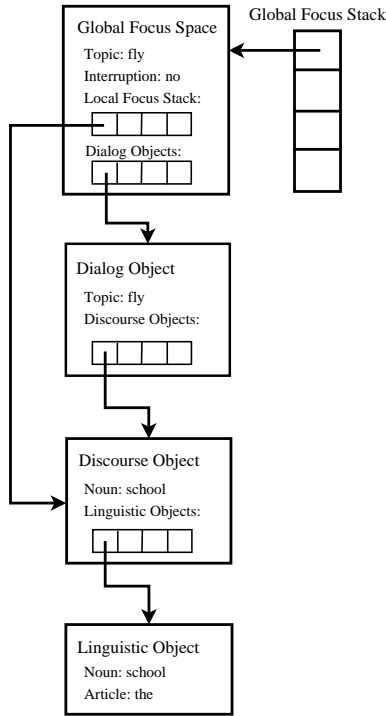


Figure 7.2: An example of a discourse model.

## 7.4 The Case Base

The actual dialogs are saved in the case base. The basic functionality is hand crafted, but more functionality is added automatically when the system is used. There are two basic case structures in the case base: *plan cases* and *plan items*.

Both the plan cases and the plan items are written in the Knowledge Machine (KM) representation language, where each case is an object in the knowledge system. The use of KM structures the definitions, and the KM

framework facilitates manipulation of the cases.

### 7.4.1 Plan Case

A plan case contains the plan for solving the problem. It consists of three parts:

- A *Problem Part* that describes which problem the case is solving.
- A *Discourse Part* that describes how the discourse model should be updated according to the problem.
- A *Plan Part* containing a plan that solves the problem. The plan consists of other plan cases or plan items that themselves are cases in the case base.

The problem part consists of the name of the case, the sequence of classifications according to the classified words in the problem utterance, and which actual words were used in this particular problem. The following is an example of the problem for the fly-to plan:

```
(a problem with
  (name (fly-to-plan))
  (words ( (:seq fly-command to-word the-word building)))
  (fly-command (fly))
  (to-word (to))
  (the-word (the))
  (building (hospital)))
```

The discourse part holds information about what has previously been said in the dialog. In the plan cases, the discourse part is usually updated with a new global focus space, a dialog object, one or several discourse objects and their corresponding linguistic objects. A new global focus space is created when a new dialog is started and the plan cases are often executed as the start of a new dialog. The discourse information corresponds to the phrase from the operator. The discourse part of the fly-to plan looks like this:



```

(a discourse with
  (discourseobjects
    ( (:seq global-focus-space dialog-object
          discourse-object linguistic-object)))
    (global-focus-space
      ((a global-focus-space with
        (function
          ("(lambda ()
            (make-global-focus-space
              'fly 'fly 'open t)))))))
      (dialog-object
        ((a dialog-object with
          (function
            ("(lambda ()
              (make-dialog-object
                'fly)))))))
        (discourse-object
          ((a discourse-object with
            (op-parameters (hospital))
            (function
              ("(lambda (hospital)
                (make-discourse-object
                  '((noun ,hospital))))))))))
          (linguistic-object
            ((a linguistic-object with
              (op-parameters (the hospital))
              (function
                ("(lambda (the hospital)
                  (make-linguistic-object
                    '((linguistic-word ,hospital)
                      (article ,the)))))))))))

```

The first lambda function is evaluated and calls the function `make-global-focus-space`. This function is a CEDERIC standard function that is defined in the core system. The arguments describe the topic of the dialog, what word was used, and the fact that the dialog is open. The dialog ob-

ject saves information about the action of this particular dialog, i.e. `fly` in this case. The discourse object and the linguistic object represent the helicopter's requested destination, which in this case is `the hospital`. They are represented as a noun to be used as a reference later on in the dialog. The linguistic object holds the information about the article used when the word was mentioned and the actual word used when the object was introduced or referred to.

The plan part of the case describes the plan, and in our fly-to example it looks like follows:

```
(a plan with
  ((:seq find-unique-reference-id get-building-position
      in-air-plan fly-to-position stop flyto-result stop)))
```

The items in the plan are names of other plan cases or plan items to be executed.

To make it easier to structure the cases written by hand, CEDERIC supports the use of subplans. A subplan does not necessarily have any words in its problem part, and the discourse part may be absent. It does always have a name and a plan part. A subplan may for example be to check if the UAV is in the air before executing a fly-to command, like the `in-air-plan` in the plan for the fly-to command shown above.

## 7.4.2 Plan Item

Plan items consist of the following parts:

- A *Problem Part* that describes which preconditions that have to be met to be able to execute the plan item.
- A *Solution Part* containing a solution to the problem in some representation.

The problem part consists of the name of the plan item, a list of preconditions that have to be met, and a goal. The goal is a description of the possible result types that are created when the solution part is executed. The preconditions and the goal are used for planning purposes. To be able to construct a plan in advance, a description of possible result types is

needed, to be used as preconditions for the next plan item in the plan. The plan item `get-reference-id` gets a building, e.g., `school`, and returns the internal identifier for every known school in the domain knowledge base. The problem part of `get-reference-id` looks like this:

```
(a problem with
  (name (get-reference-id))
  (precond
    ((a precondition with
      (attributes ([:seq building]))
      (building (known)))
    (goal
      ((a goal with
        (attributes ([:seq several-buildings
                     unique-building
                     no-building]))
        (several-buildings (known))
        (unique-building (known))
        (no-building (known))) )))
```

It needs a building as a precondition and it returns either several identifiers that suit the description, a unique identifier, or none at all. Which result is actually returned from the solution function is of course not known until execution time, but the planner can take different possible outcomes into account.

The solution part consists of lists of parameters and a solution function. The solution function can for example construct a message to be sent to the helicopter control system, or perform some internal computations. The solution function returns both the result type, which is one of the possible types listed in the goal of the problem part, and the actual result value. The result type is used if additional planning needs to be performed. The solution part of the `get-reference-id` plan item looks like this:

```
(a solution with
  (parameters (hospital))
  (solution-fn
    ("(lambda (noun)
```

```

(defun get-id (noun)
  (km '#$(every ,NOUN)))
(let ((ids (get-id noun)))
  (cond
    ((endp ids)
     '((no-building))(no-id)))
    ((= 1 (length ids))
     '(((unique-building))(unique-id ,(car ids))))
    (t '(((several-buildings))(several-ids ,ids))))))

```

The solution function gets the word `hospital` or a similar word, finds the internal id of every such noun and returns them. It also returns `no-building`, `unique-building`, or `several-buildings` depending in how many matching identifiers were found.

An example of a solution that returns a message to the helicopter control system is the solution to the plan item `fly-to-position`:

```

(a solution with
  (computed-parameters (position in-air))
  (solution-fn
    ("(lambda (position in-air)
      (let ((request-id (create-request-id)))
        '(((fly-request) ,request-id)
          (say-out
            (progn
              (say-to-operator
                (quote (outphrase :content (ok))))
              (send-to-agent
                (quote
                  (do!
                    :content
                    (fly-to-position ,(car position)
                                     ,(cadr position))
                    :request-identifier ,request-id))))))))))"
    )))

```

This solution function takes two computed parameters as argument, which means values that have been computed by other plan items earlier in the plan. It returns the result type `fly-request`. It also returns a message to be evaluated later which sends an `ok` to be told to the operator and a `fly-to-position` command to be sent to the agent. The request identifier is a unique identifier to be sent to the robotic control system. The control system sends the same identifier back with every response to that particular request, so that the dialog system knows which message is a response to which request.

## 7.5 Case-Base Manager

The case-base manager is the engine of CEDERIC. It manages different ongoing dialogs and executes the best fitting plan item. If there are open and ongoing dialogs, the new input is first seen as a continuation of one of the open dialogs. If it does not fit as a continuation, it is seen as a start of a new dialog, and a suiting similar plan case is searched and the plan is executed. The case-base manager is written in Lisp, which is also the implementation language for the Knowledge Machine. This makes the code uniform and easy to follow.

### 7.5.1 Dialog Handling

When a new sentence arrives from the operator or agent, it could be one of the following cases:

- The operator or agent continues the current dialog, possibly by the start of a subdialog.
- The operator or agent returns to an older non-completed dialog that is not presently in focus.
- The operator starts a new dialog, possibly without ending the recent dialog properly.

As mentioned in section 7.3, the global focus stack keeps track of the open global focus spaces. The global focus spaces are linked to the plan corresponding to the dialog via a *plan agenda*. When new input from the

operator enters the system, the case-base manager searches the global focus stack for open dialogs. If such a dialog is found, the system tries to execute the first plan item in the plan, using the discourse found in the global space, i.e., the system believes the input is a continuation of the dialog in focus. If the input does not match the parameters for the execution of the plan item, the next open dialog is tried, etc. If the input does not match any open dialog, the system handles the input as the start of a new dialog, searches the case base for a suiting case, and creates a new discourse.

If, on the other hand, there is a message from the robot that does not match the present discourse in focus, CEDERIC has to take topic management into consideration. A report of a result of a performed command shall for example not be mentioned right away if the operator waits for an answer to a question. CEDERIC decides what to do by investigating the current content of the global focus space, and checks if it is ok to interrupt in the present discourse or not. If it is ok to interrupt, the same algorithm as the one for an utterance from the operator is performed, but if it is not, the message is put in a queue and is evaluated as soon as it is ok to interrupt, or the present dialog has been closed. Figure 7.3 gives an example of a dialog with both topic changes and topic management, where the execution of a message from the robot is delayed until the ongoing subdialog is closed.

Dialog handling allows dialog topic switches, where the operator may start a new dialog before ending an old one, or return to an old dialog, without confusing the system.

### 7.5.2 Syntactic Categorization of Words

The utterance from the operator enters the system as a sentence in text format. Each word in the utterance may be categorized using a lexicon. The lexicon entries are written in the KM language, similar to the case base. An entry could for example look like follows:

```
(school has
  (instance-of (building))
  (attributes (:seq plural)))
```

---

Operator: Fly to the school.  
 CEDERIC: I have two schools to choose between.  
           Which one do you mean?  
 Operator: Take off.  
 CEDERIC: Ok.  
 Operator: Which can I choose between.

*CEDERIC gets a message from the robot saying that the action take off has been successfully completed*

CEDERIC: You can choose between the one on Harborroad and the one on Mainstreet.  
 Operator: Fly to the hospital.  
 CEDERIC: Ok.  
 CEDERIC: I have taken off now.  
 Operator: What is your altitude?  
 CEDERIC: It is 20 meters.  
 CEDERIC: I am at the hospital now.

---

Figure 7.3: An example of dialog topic changes and topic management between the operator and CEDERIC.

```
(plural (schools))
(weight (2))
```

where `school` is modeled to be an instance of the category `building`. It also has the attribute `plural` which is set to `schools`, and a `weight` which is set to 2. The `weight` is used in the similarity function and is an indication of how important the word is for the comprehension of an utterance. A word may be ambiguous and belong to several different categories depending on the semantics of the utterance. In that case, the word is categorized with all the matching categories and the similarity function decides which meaning to use. A word that is not present in the lexicon obtains the category `no_category`.

### 7.5.3 Case Retrieval

In the retrieval phase, the utterance is compared to the problem part of each plan case and a similarity value is computed. The cases are then prioritized using the similarity value. The similarity value is based on how well each word in the utterance matches the words in the problem formulation and the weight of each word according to the lexicon. The case retrieval phase is only executed if the input utterance is a start of a new dialog, i.e. there are no old or ongoing dialogs where the utterance fits as a continuation.

To be more specific, the similarity value for each plan case is computed using the following formula, where  $w_c$  means the sequence of words in the problem part of the case and  $w_i$  means the sequence of words in the input from the operator:

$$\textit{Similarity} = \textit{points}(w_c, w_i) - \textit{uncovering}(w_c, w_i)$$

where

$$\textit{points}(w_c, w_i) = \sum_{w_x \in w_c} \sum_{w_y \in w_i} \textit{class - value}(w_x, w_y) \times \textit{word - weight}(w_y)$$

$$\textit{class - value}(w_x, w_y) = \begin{cases} 3 & \text{if the words are equal} \\ 2 & \text{if the words are classification similar} \\ 0 & \text{otherwise} \end{cases}$$

$$\textit{uncovering}(w_c, w_i) = |\textit{length}(w_c) - \textit{length}(w_i)|$$

Two words,  $w_1$  and  $w_2$ , are *classification similar* if either  $w_1$  is equal to the classification of  $w_2$ , the classification of  $w_1$  is equal to  $w_2$ , or the classification of  $w_1$  is equal to the classification of  $w_2$ .

When two cases have the same similarity-value they are subprioritized using the *uncovering* function which indicates how much the sequences differ in length. A low value on the *uncovering* function is prioritized.

When the cases are ranked in order of priority, the case with the highest priority is chosen unless it does not have lower similarity value than a threshold. In that case, no similar case was found.



The order of the words in the input utterance is respected by searching for a subsequence matching the input utterance in the case words. If no similar case is found, the cases are reprioritized in a manner where the order of the input words does not matter. In this way cases with equal order of words are selected in the first place, but the system can also understand phrases where the words are out of order.

The plan of the most similar plan case is retrieved and validated using the input utterance and the precondition and goal in the problem part of each plan item. See section 7.4.2 for a description of preconditions and goals. The categories of the words in the user utterance is used to search for matching preconditions for the first plan item in the plan. If the preconditions are satisfied, i.e. the words in the precondition are present in the user utterance, the goal of the plan item is added as another potential precondition in case items further on in the plan. The whole plan is checked and if all the preconditions for every plan case is found, the plan can be executed. On the other hand, if plan faults are found, the plan is sent to a plan repair unit, which tries to repair the plan using other plan cases. The plan repair unit is described in detail in section 7.5.5.

#### 7.5.4 Case Reuse

When the input is a beginning of a new dialog and a similar suitable plan case is found, the discourse part of the plan case is executed. The discourse part is a function which describes how the discourse should be updated according to the new input. When it is a start of a new dialog, the discourse part creates a new discourse and initializes it with the input information. An explanation of the discourse part in a plan case can be found in section 7.4.1. When the discourse in the plan case is updated, the plan is executed.

A plan may consist of plan items, plan cases and *stop* items. Subplans are modeled as plan cases within a plan. Before execution, the plan case has to be evaluated and exchanged for the plan of the plan case. Stop items are special plan items because they do not have a corresponding case. When the system encounters a stop item, the plan execution stops and information to the robotic control system or the speech generator can be delivered.

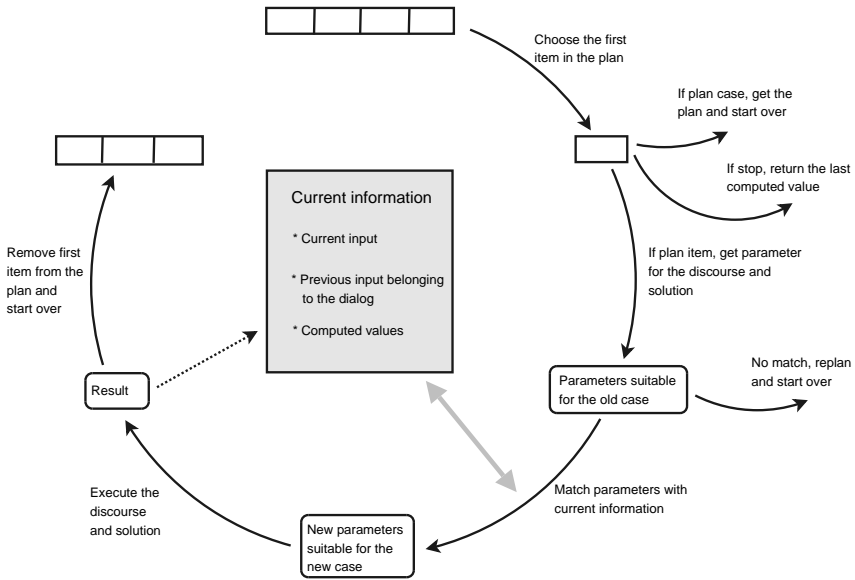


Figure 7.4: Schematic view of the reuse phase.

Regardless of whether the input is a start of a new dialog, or a continuation of an open dialog, the plan is executed in the same manner, as shown in Figure 7.4. For each plan item:

- Find values similar to the parameters of the discourse function and the solution function of the case item.
- Execute the discourse function that updates the discourse according to the information in the case.
- Execute the solution function that returns the result of the case.
- Save the result and execute the next item in the plan.

The parameters of the discourse function and the result function are listed in the discourse part and the solution part of the plan item, respectively (see section 7.4.2 for examples). Parameters can be of three different types:

*Input Parameters.* These are parameters whose values are part of the latest input or previous input originating from the operator or the robotic control system. The previous input has to belong to the current dialog.

*Computed Parameters.* Results of previously executed solution functions belonging to plan items in the current dialog are called computed values, hence computed parameters are parameters which values are members of the computed values.

*Discourse Parameters.* This is usually the discourse object currently in focus.

The input parameters listed in the case are the values of the parameters that suited that particular case. The new problem may be slightly different because it is similar and not necessarily identical. To make sure the discourse function and the solution function is adapted to the new problem, other parameter values have to be sent to the functions. Suitable input parameter values are found using the same classification similarity function as in the retrieval phase described in the previous section. For example if the input parameter is **red**, then **green** is a valid substitute, because both of the words are members of the category **color**.

The computed values are tuples of the form **<category value>**. The computed values list the category of the parameter value, and hence the value can be directly found in the set of computed values by searching for the identical category. The values for the discourse parameters are obtained by identifying the discourse structure of the discourse parameter and retrieving the corresponding discourse structure for the discourse of the new problem. The discourse parameters are used when the meaning of an utterance can not be found without information about earlier utterances in the dialog. An example of such a dialog is given in Figure 7.5, where the the word **one** refers to the word **school** mentioned earlier.

---

Operator: Fly to the school.  
CEDERIC: I have several schools to choose between.  
          Which one do you mean?  
Operator: The red one.  
CEDERIC: Ok.  
CEDERIC: I am at the school now

---

Figure 7.5: An example of a subdialog using a clarifying question.

If suitable values for the parameters of the discourse and solution functions can not be obtained, the execution of the plan item has failed and the plan has to be revised. Replanning in this and other situations is discussed in the next section. However, if suitable values are found, the discourse function is executed first, followed by the execution of the solution function. The discourse function manipulates the discourse model by adding more information to it. It does not return any value. The solution function produces a result, on the other hand. The result is made up of two parts, namely a result type and the actual result. The result type is of the same type as the preconditions and goals in the problem part of plan items. The goal lists the possible result types of a particular plan item, and the result type in the result is the actual result type that was produced by the plan item for this particular input. The result type is used if the plan fails later on in the execution, so that replanning is necessary. The actual result is saved as a computed value and can be used as a computed parameter value for plan items later on in the plan. It may also be information to be sent to the operator or the robotic control system. If the system encounters a stop item in the plan, the execution stops and the last computed value is examined. If the value is labeled *say-out*, the value is sent either to the robotic control system or to the speech generator. The system examines it further and decides how to handle this particular value. The value can also be a compound result, consisting of both an utterance to be told to the operator and a command to be sent to the robot.

### 7.5.5 Replanning

Replanning is used in two different phases in CEDERIC:

- When there is a plan fault in the plan checking process of the most similar plan case, i.e. when the preconditions of the plan items in the plan are not met.
- When there is a plan fault in the execution of a plan, i.e. there is no suitable value for one or more parameters.

The same replanning routine, based on case-based planning methods, is used in both cases. In CBP, different plans are combined to form a new plan that can serve as a solution to a new problem. In CEDERIC, two plans can be combined to form a new plan, where one of the original plans works as the foundation on which the new plan is constructed. The preconditions and goals in the plan items are used in the replanning process to make sure the newly constructed plan is a valid one.

When a plan fault is encountered, the goals of the previous plan items in the plan, or the result types if it was an execution plan fault, are sent to the replanning routine together with the rest of the plan, i.e. the plan where the plan item that caused the fault is the first item. Every plan item in the case base that can be executed given the current result types or goals is found, e.g., every valid next action and every plan containing these plan items are listed. Then the system tries to merge sequences of these repair candidate plans with the original plan. If a valid merge is found, it is linked to the already checked or executed part of the plan. If no valid plan is found, the first item in the original plan is removed and the algorithm tries again to find new, partly matching plans to be used to repair the original plan. In this manner, parts of the original plan is switched against parts of another plan. The newly found plan sequence is then linked back to the original plan. Given the already checked or executed plan items, the newly created plan consists of a sequence from the foundation plan, a sequence from a repair candidate plan, and then again a sequence from the foundation plan. Figure 7.6 shows a schematic illustration of such an example. This is very useful when the system realizes that all preconditions that have to be fulfilled to perform a command from the operator, are not fulfilled. Using

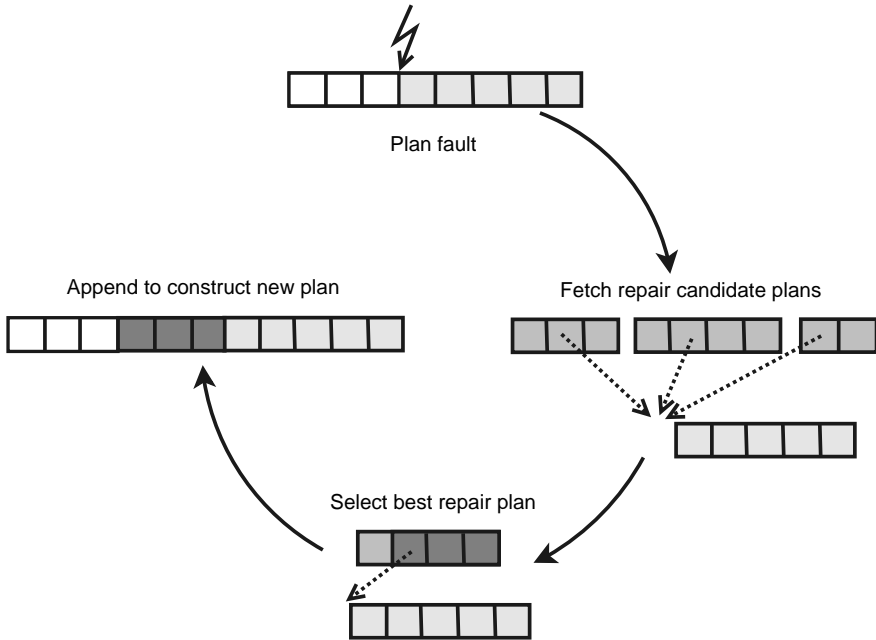


Figure 7.6: A schematic view of replanning in CEDERIC.

replanning, CEDERIC can automatically perform some actions to be in a state that fulfills the preconditions. An example is when the operator wants the helicopter to perform a fly command, but the helicopter has not taken off yet. Then CEDERIC can automatically perform a take off and then execute the operator's command. In more detail, the original plan for the utterance `Fly to the hospital` looks like follows:

```
(find-unique-reference-id get-building-position in-air-plan
 fly-to-position stop flyto-result stop)
```

where `in-air-plan` looks like:

```
(in-air-question stop in-air-answer in-air-result)
```

`fly-to-position` has `in-air` as a precondition and also as a computed parameter, which `in-air-plan` returns if the UAV has an altitude bigger than zero. However, if the UAV is on the ground, `in-air-plan` returns `not-in-air`. Hence, when the plan is executed, a plan fault occurs when the UAV is on the ground, because `fly-to-position` does not get values for all the parameters. CEDERIC searches the case base and finds a set of plan items which can be executed instead of `fly-to-position`. The plans where they occur are examined and the plan

```
(take-off stop take-off-result stop)
```

is a valid continuation from the current point of execution and it has `in-air` as a goal. Hence, the new plan looks like:

```
(find-unique-reference-id get-building-position in-air-plan  
  take-off stop take-off-result stop fly-to-position  
  stop flyto-result stop)
```

which solves the problem.

The original plan may also be totally replaced by the new plan, where the already checked or executed plan items are linked to the newly found plan, which never returns to the original plan again. This is useful when the system needs to combine the first part of one plan with the last part of another one to be able to solve a problem.

The revised plan may again obtain a plan fault when executed, and it is then repaired using the same replanning routine, but the system has to make sure it does not repair the plan with a solution that has been tested and failed earlier. To avoid such cycles, the new plan is compared to the execution history, and the new plan has to be unique in some sense to be a promising repair plan.

When a replanning sequence has occurred, the discourse in the original plan case may not suit the new plan. To solve this, the plan cases used to build the new plan are saved and examined for a suitable discourse part.

The greatest advantage of the replanning routine is the simplicity. It is easy to understand and to implement, and it works well in small domains. However, it has some major disadvantages. As the case base grows larger, more candidate plans are found and need to be checked, hence the

replanning procedure turns out to be very time consuming. Better indexes and pruning facilities may speed up the replanning procedure considerably. Another disadvantage is the lack of an overall goal of each plan case. The replanning routine, as described above, may find a plan that is valid with respect to the preconditions in the plan items, but the overall plan may not solve the right problem. This disadvantage may be acceptable in a learning system, where new solutions should be tried and rejected if not good enough. However, with a more complex structure including plan goals, the speed and correctness of the replanning routine can be increased.

### 7.5.6 Case Retention

When a problem is not identical to the problem part of the plan case that was used for solving it, and the execution of the problem was successful, the problem and solution are saved as a new case in the case base. The new cases increase the case base and the new experiences can be used to solve other problems in the future. A new problem and solution can differ from the cases used to solve it if one or more of the following has happened:

- One or more of the words in the input are not identical to the words in the problem part of the plan case. They are only classification similar.
- One or more of the words in the input are added or deleted from the words in the plan case, but the plans are identical to each other.
- The plan has been replanned in the plan check phase.
- The plan has been replanned in the plan execution phase.

In the first case, the executed plan is identical to the one in the plan case, but the problem part of the plan case and the parameters in the plan items may be different. A new plan case, with a new name and a problem part that corresponds to the input is created. The plan items in the plan are checked, and new plan items with new names are created if the input parameters have to be changed to correspond to the input. Finally, the plan is updated with the names of the newly created plan items. An example



where this occurs is if the case with the problem **Fly to the hospital** is used to solve the problem **Fly to the school**.

In the second case, a new plan case with a unique name is created, where the problem part corresponds to the words in the input. Since the plan is equal, no new plan items need to be created. An example of this case is if the case with the problem **Fly to the hospital** is used to solve the problem **Fly to hospital**.

The last two cases are slightly more complicated. In the third case, the input and the problem part of the plan case rarely have a high similarity score because if they had, no replanning would have occurred. A new plan case is created in the same manner as for the second case, but the new plan is added to the new plan case as well. If some of the parameters in the plan items are only classification similar, new plan items are created and the plan is updated with the names of the new items. An example is where the cases with the problems **Fly to the hospital** and **What is the position of the red car** is used to solve the problem **Fly to the red car**.

In the fourth case, the problem part of the plan case is often identical or classification similar to the input, but the situation is not identical. If the problem part was not identical or classification similar, a replanning sequence would have taken place in the plan check phase and not in the plan execution phase. Let us look at an example where the system knows about two schools and one hospital. When the operator says **fly to the school**, a case where the problem is **fly to the hospital** is used. However, the plan for this case can only be used for uniquely described buildings, and it will fail when executed with the word **school**. This plan fault can not be identified before the execution of the plan is started, because one part of the plan is to find which building the operator refers to, hence the failure will only be recognized during execution. A new plan case is created, with a unique name and the new plan. If the input is identical to the problem part of the plan case, nothing more is done, but if it is only classification similar, the problem part of the plan case and new plan items are created in the same manner as described for the first case above.

The four cases presented above may not only happen in isolation, but a combination is also possible and can be handled by the system.

---

Operator: Fly to the supermarket.  
CEDERIC: I do not know what a supermarket is.  
Is it a building?  
Operator: Yes.  
CEDERIC: Can you give me a reference object near the supermarket so I can start by flying to it?  
Operator: It is near the red hospital.  
CEDERIC: Ok, I will start by flying to the hospital and then I will ask for more guidance.  
CEDERIC: I am at the hospital now.  
CEDERIC: In which direction shall I fly to look for the supermarket and what characteristics does it have?  
Operator: Fly eastwards and look for a blue building.  
CEDERIC: I start fly and look for the supermarket.  
I will stop when I see it.  
CEDERIC: I have found the supermarket now and hover over it.

---

Figure 7.7: An example of a teaching situation between the operator and CEDERIC.

## 7.6 Learning from Explanation

CEDERIC can ask the operator for guidance if the input contains a word that can not be syntactically categorized. If no category can be found for a word, it is labeled with the category `no_category`. To be able to catch those `no_category` utterances, CEDERIC needs to have cases that match both the sentence where the `no_category` was found and the `no_category` itself. By providing CEDERIC with such cases, it can ask the operator questions that makes him or her explain how CEDERIC should react. Assume we have a case where the problem part is built on the utterance `fly to the pizzeria`, where the word `pizzeria` is classified

with the `no_category` classification. When the utterance `fly to the supermarket`, where the word `supermarket` is unknown and classified with `no_category` enters the system, it is similar to the pizzeria plan case and its plan is executed. Depending on the answers from the operator, the result can be the dialog in Figure 7.7.

After the execution of this dialog, the lexicon is extended to also contain the word `supermarket` which is a `building`. The newly gained information such as the color and the category of the supermarket will be saved in the domain knowledge as well. The next time the operator wants the robot to fly to the supermarket, the sentence will be fully classified and the case that matches such a sentence will provide a correct solution.

The dialog shown above where the operator guides the UAV to an unknown building can be used with any word that the operator wants to use as an identifier for a particular building. This particular plan case is however not general enough to handle other types of words which are not buildings. They can be handled analogously by using other plan cases describing the subdialog suitable to teach CEDERIC how to react properly on them.

## 7.7 An Example

This section gives an example of a small case base and dialogs that can be understood by the system using this case base. This toy example is much smaller than the case base used in CEDERIC, in terms of number of plan cases and plan items, but it illustrates the main features nicely.

Assume the case base contains two plan cases, with the following problem phrases:

- Fly to the hospital
- What is the position of the car

The first one is called `fly-to-plan` and the plan for it was given in section 7.5.5, but is recapitulated here as well:

```
(find-unique-reference-id get-building-position in-air-plan
 fly-to-position stop flyto-result stop)
```

where `in-air-plan` is a plan case itself, but with no problem phrase, and its plan looks as follows:

```
(in-air-question stop in-air-answer in-air-result)
```

The second one is called `where-vehicle` and the plan for it looks as follows:

```
(vehicle-position-question stop vehicle-position-answer  
  say-position stop)
```

In addition to these plan cases and plan items, the system has a lexicon with the following structure:

```
(hospital has  
  (instance-of (building))  
  (attributes ([:seq plural]))  
  (plural (hospitals))  
  (weight (2)))
```

```
(school has  
  (instance-of (building))  
  (attributes ([:seq plural]))  
  (plural (schools))  
  (weight (2)))
```

```
(car has  
  (instance-of (vehicle))  
  (attributes ([:seq plural]))  
  (plural (cars))  
  (weight (2)))
```

```
(truck has  
  (instance-of (vehicle))  
  (attributes ([:seq plural]))  
  (plural (trucks))  
  (weight (2)))
```

```
(fly has  
  (instance-of (fly-command))  
  (weight (2)))
```

```
(go has
  (instance-of (fly-command))
  (weight (2)))
```

accompanied by structures for the words **what**, **to**, **the**, etc.

Given these plan cases, plan items, and the lexicon, which phrases can be evaluated by the system? First of all, phrases that are identical to the problem phrases stated above can of course be evaluated. Moreover, phrases where one word has been exchanged for another word belonging to the same category, can be evaluated. In this case, the following phrases are accepted:

- Fly to the school
- Go to the school
- What is the position of the truck

The plan cases also give an opportunity for replanning. The solution for the phrase **Fly to the truck** can be obtained by combining parts of the plans from the plan cases. The similarity function ranks **fly-to-plan** as the most similar plan. However, it does not fulfill the requirements of the preconditions, because the first plan item **get-reference-id** expects a building, hence replanning is initiated. The replanning routine starts by trying to find another plan or subpart of a plan that can provide a building for the **get-reference-id** plan item. No such plan or subpart of a plan is found, hence the replanning routine tries to find a plan or subpart of a plan that can connect to the next item in the plan, that is **get-building-position**. No such plan or subpart of a plan is found either, because **get-building-position** needs a reference id of a building. The next plan item in the plan is a subplan called **in-air-plan**. This subplan has no preconditions and can always be executed, hence the replanning routine checks the plan from **in-air-plan** and onwards to see if the plan can be executed. The plan checking fails when it checks the plan item **fly-to-position** because it needs a position to fly to. Again the replanning routine kicks in and searches for a plan or subplan that can provide the plan item with a position. This time, the plan case **where-vehicle** is useful. The subpart

(vehicle-position-question stop vehicle-position-answer)

of the plan returns a position of a vehicle, which works as a precondition for the plan item `fly-to-position`. It needs a vehicle as a precondition, and this is provided in the original phrase from the operator. Hence, the plan

```
(in-air-plan vehicle-position-question stop
  vehicle-position-answer fly-to-position stop flyto-result
  stop)
```

is checked and executed. The information about where to fly is stored in the discourse model, hence the plan item `flyto-result`, that tells the operator that the helicopter has completed flying to a particular place, says `I am at the car now`, which is perfectly correct. Hence, the system manages to execute the commands

- Fly to the car
- Fly to the truck

as well. The commands

- What is the position of the school
- What is the position of the hospital

are solved in a similar manner using replanning.

In addition to the phrases already listed, the system also handles phrases where some of the words have been left out. The phrases

- Fly to hospital/school
- Fly hospital/school
- What is position car/truck
- What position car/truck

are evaluated correctly, but phrases such as `Fly` and `Position car` are too ambiguous and lack too much information for the system to make an interpretation of them.

The phrase `Fly truck` does not work before the phrase `Fly to the truck` has been executed. That is because `Fly truck` is not considered similar enough to `Fly to the hospital`, hence the replanning routine is not initialized. However, after `Fly to the truck` has been executed, a new plan case is stored in the case base, with `Fly to the truck` as the problem phrase and the newly planned plan as the plan. This new plan case is similar enough to `Fly truck`, and the plan is executed.

With this case base consisting of three plan cases, where one of them is a subplan, and ten plan items, four different types of commands can be executed in at least twelve different ways. Notice again, that this was a mini-example for the purpose of explanation, and that the actual case base in the CEDERIC system is significantly larger.





# Chapter 8

## Tests and Results

### 8.1 Development Tests

It is important to test the system regularly during the development phase, to be able to find design faults and bugs related to new implementations. The test scripts also carry information about the status and the performance of the system. As new features and cases are implemented, the test scripts are modified and new test cases are added to the test suite.

CEDERIC has been tested regularly during the implementation phase. At most the case base contained 51 handwritten plan cases and 65 handwritten plan items as the basic case base, and more cases were added automatically during the execution of the tests. The test suite contains test cases that test several aspects of the system, namely:

- Cases that are identical to the plan cases in the case base.
- Cases that test the similarity function.
- Cases that test dialog handling.
- Cases that test overall flexibility of the system.
- Cases that test the replanning functionality.

The first four types of tests measure the performance of individual cases, whereas the test results of the replanning functionality depend on the environment where the test is executed, i.e. the result of a replanning session depends on the cases in the case base. A replanning case that creates the optimal solution when executed using one set of cases in the case base may not create the optimal solution using another case base, due to inference of other cases in the replanning process. To ensure correct behavior in a replanning situation, regardless of the other interfering cases, the test module for CEDERIC selects the execution order of the test cases randomly with a few exceptions. The exceptions are those test cases that test learning from explanation and then test if the newly learnt information is stored correctly in the system. In such test cases, the order of the evaluation is important and the learning case has to be executed before the test that checks the learned information is executed.

The automatic test scripts simulate both the operator and the robot, using prestored static messages that are sent to CEDERIC, hence CEDERIC is the only system in the loop. An example of a test case is shown in Figure 8.1. The information labeled CBR-OP and CBR-HELI is fed into CEDERIC from the test routines. The result from CEDERIC is compared to the result in the test, and if equal, the next line of the test case is evaluated in CEDERIC. If the answers produced from CEDERIC are not equal to the answers in the test, the test is marked as failed. The reason for not testing the system together with the graphical user interface and the simulator or robot is partly because it is convenient to test CEDERIC separately, and partly because it is impossible to perform automatic testing with prestored solutions for tests performed in random order. The answer to status questions such as `what is the position/altitude/velocity` depends on the execution history of the robot, and if the execution history is randomly chosen, static test scripts can not identify a success or failure of a test case.

Regular testing of the system has turned out to be a very helpful tool to find both design faults and bugs in the system. Problems with early versions of the similarity function were easily detected and the function could be rewritten according to the particular problem. The problems with the replanning routine described in section 7.5.5, were also efficiently detected using the test scripts.

---

```

(test30
  (CBR-OP '(SHOW ME THE FRONT FACADE OF THE CHURCH))
  (CHOOSE-MESSCONS '(VALUE? :CONTENT (HELPOS)
                        :REQUEST-IDENTIFIER )
                    *IMESS-CONS-LIST*)
  (CBR-HELI '(ANSWER :QUERY (HELPOS) :ANSWER (10 170 19)
                :REQUEST-IDENTIFIER ))
  (CHOOSE-MESSCONS '(VALUE? :CONTENT (HELALT)
                        :REQUEST-IDENTIFIER )
                    *IMESS-CONS-LIST*)
  (CBR-HELI '(ANSWER :QUERY (HELALT) :ANSWER 10
                :REQUEST-IDENTIFIER ))
  (CHOOSE-MESSCONS
    '(DO! :CONTENT (ROTATECAM -90 26)
          :REQUEST-IDENTIFIER )
    *IMESS-CONS-LIST*)
  (CBR-HELI '(REPORT :STAGE CONCLUDING-ACTION
                    :REQUEST-IDENTIFIER ))
  (ADDSAYLIST '(OUTPHRASE :CONTENT (IS THIS VIEW OK))))
)

```

---

Figure 8.1: An example of a test from the test scripts.

## 8.2 Simulation Tests

As a complement to the development tests, CEDERIC is also tested together with the SGUI user interface against the Hazard simulator. Chapter 5 described SGUI, and section 5.4 in particular described the test procedure and the Hazard simulator. Figure 8.2 and Figure 8.3 show examples of what the simulator and the SGUI look like when connected to CEDERIC in a simulation test.

CEDERIC has been tested manually using both spoken language and written text. The manually performed tests are more realistic than the automatic development tests and expose the system to spontaneous errors

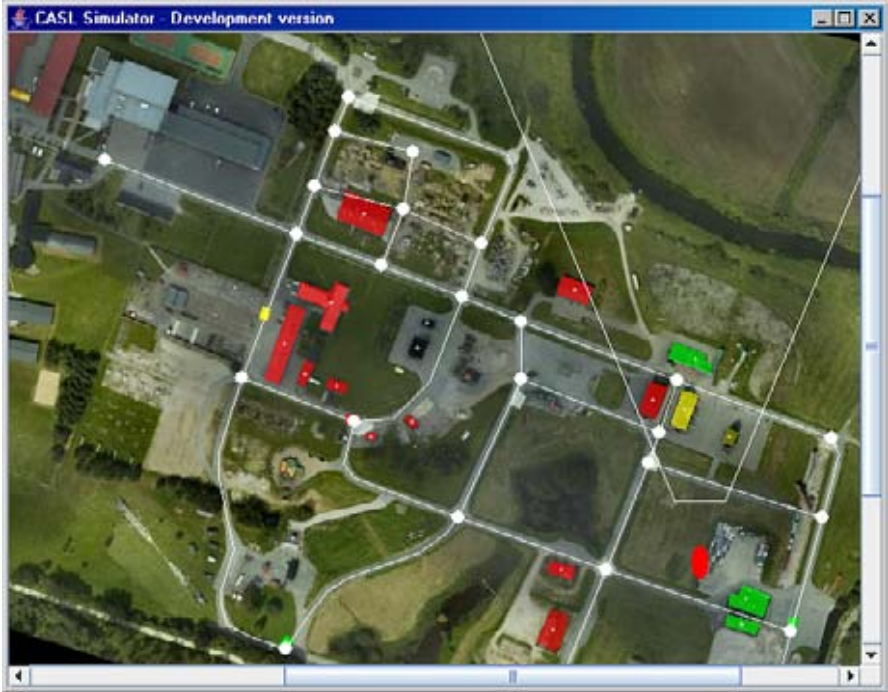


Figure 8.2: The simulator for simulation tests where the camera view is indicated as a triangle like area.

such as speech recognition and typing errors. This type of testing also tests the connections between the subsystems.

The simulation tests have been performed both as loosely structured tests where the operator improvises during the tests, and as more formal tests where the operator works on the basis of a scenario. The benefit of improvised tests is the ability to test different dialogs and to provoke the system with unexpected input. Structured tests using a scenario are intended to be closer to reality. The scenario presents the operator with a



Figure 8.3: The SGUI in the middle of a simulation test.

specific situation and a task to be performed, and it is up to the operator and the system to accomplish the task. The following two subsections will present two different scenarios together with a dialog example and comments on the result.

### 8.2.1 Scenario I

The following scenario serves as a starting point for the operator:

Operator: Fly church.  
CEDERIC: Ok I am starting to fly.  
CEDERIC: I am at the church now.

*The operator gets the new report.*

Operator: Rotate camera ninety degrees right.  
CEDERIC: I am moving the camera.  
CEDERIC: I have rotated the camera ninety degrees right now.  
Operator: Fly to the hospital.  
CEDERIC: Ok I am starting to fly.  
Operator: Video ten minutes.  
CEDERIC: I do not know what you mean with the phrase video ten minutes.  
Operator: Play video ten minutes.  
CEDERIC: Here it comes.  
CEDERIC: I am at the hospital now.

---

Figure 8.4: An example of dialog based on a scenario I.

*There have been reports about a drunk driver in a red car, last seen close to the church. Your mission is to use the UAV to search for the car. When the UAV is near the church, you got new information that the car has been seen several times near the hospital during the last ten minutes. Direct the camera towards the nearby hospital and try to get an observation of the car. Then fly to the hospital and request the video recorded ten minutes ago. The UAV is in the air near the base at the start of the mission.*

The simulation of the world in scenario I is the same as the one shown in Figure 8.2. The operator has information about the different categories of the buildings. A tested dialog for the scenario is shown in Figure 8.4. In this dialog, the CBR similarity function is tested as well as the actions themselves. The operator starts by saying **Fly church** instead of **Fly to the church** as stored in the case for the action. CEDERIC interprets it

correctly as **Fly to the church** and executes the command. The next problem is when the operator tries to request the video. The command **Video ten minutes** is not similar enough to **Show me the video from ten minutes ago**, as stored in the case base, and CEDERIC can not interpret the utterance. The operator tries again, but now with the phrase **Play video ten minutes**. The word **play** is in the lexicon and has the same category as **show**, hence the similarity is enough for CEDERIC to correctly interpret the phrase. The decision not to interpret the phrase **Video ten minutes** as a request to show the video recorded ten minutes ago seems reasonable, due to the vagueness of the phrase. On the other hand, to interpret the phrase **Play video ten minutes** as a video request seems more reasonable because of the word **play** that indicates what operation the operator wants to do with the video.

## 8.2.2 Scenario II

The second scenario looks like follows:

*There are reasons to believe that a group of terrorists are located in a small airfield on the Swedish countryside. Your mission is to use the UAV to search the area. Let the UAV return to the base if any signs of activity are detected. The current position of the UAV is on the ground on the home base.*

The corresponding dialog can be found in Figure 8.5. The dialog contains several difficulties that include replanning. The first occurrence is when the operator commands the UAV to fly to the northern side of the buildings, even though it is still on the ground and has not taken off yet. The **fly-to-plan** plan case checks the altitude of the UAV before it is executed and if the altitude is too low, CEDERIC has to find a way to increase the altitude. In this case it is done by performing a take off before the actual flight is started. When the operator wants to be shown the front facade of the main building, the UAV does not have a good angle from the current position, hence it first has to fly to a better position and then turn the camera to be able to show the front facade.

---

Operator: Your main goal today is to look for people and vehicles.  
CEDERIC: Ok I will report every observation of people and vehicles.  
Operator: Fly to the northern side of the buildings.  
CEDERIC: Ok I am taking off.  
CEDERIC: I am starting to fly.  
CEDERIC: I am at the northern side.  
Operator: Show me the front facade of the main building.  
CEDERIC: I am starting to fly.  
CEDERIC: Is this view ok?  
Operator: Climb ten meters.  
CEDERIC: Ok I have ascended ten meters.  
Operator: Rotate the camera forty five degrees right.  
CEDERIC: I have rotated the camera forty five degrees right.  
Operator: Fly to the hangar.  
CEDERIC: I am starting to fly.  
CEDERIC: I see a red car.  
Operator: Hover.  
CEDERIC: I am hovering.  
Operator: Fly to the base.  
CEDERIC: I am starting to fly.  
CEDERIC: I am at the base.

---

Figure 8.5: An example of dialog based on a scenario II.

### 8.2.3 Results

Both the development tests and the simulation tests have been important tools to test the functionality of the system. The simulation tests show that the integration with SGUI and the simulator works correctly and CEDERIC behaves as intended. Several aspects of CEDERIC have been tested in the development tests and the simulation tests, such as the behavior of the similarity function, the overall flexibility of the system, the replanning routine, nested dialogs, subdialogs and how well the system copes with



speech recognition errors.

The similarity function has been shown to function to satisfaction. The design or choice of an optimal similarity function is however a somewhat subjective matter, and the balance between intelligently interpreting a phrase and over interpreting it is hard. CEDERIC seems rather well balanced, but if an acceptance dialog where CEDERIC asks for acceptance for an interpretation is developed, as in `I do not quite understand but do you mean fly to the hospital?`, then CEDERIC may be allowed to over interpret to a larger extent.

The overall flexibility of the system has been tested by implementing different kinds of dialogs with various structures. It is easy to implement a command such as `fly to the hospital`, but do the system and the similarity function manage other types of commands? The command `fly home to the vicar of the white church` was implemented to test this. The result of the phrase is that CEDERIC finds the building where the vicar lives and flies there. The flexibility of the problem solving features and the similarity function were then tested using the phrase `fly to the janitor of the white church`, i.e. where the intended building is switched, but the building mentioned in the phrase is the same. CEDERIC solves the problem successfully and flies to the janitor's house.

The replanning routine has some design flaws which were described in section 7.5.5. The most notable problem is the execution time. It sometimes takes as much as 60 seconds for the system to find a solution to a replanning problem in a case base with about 60 plan cases, but in rare cases, it takes up to two minutes. This is not acceptable in a real situation, but as CEDERIC is a test version and a proof of concept, it should be acceptable. According to the problem formulation in section 6.2, the question is how a machine learning and problem solving system that uses the knowledge available, can be constructed. How it can be best solved is the next step. It should be possible to rewrite the replanning routine used in CEDERIC to make it more time effective, but as the case base grows larger, the problem will remain. An idea is to use better indexes and pruning facilities in addition to plan goals in each plan case which will guide and inform the search and make it more time efficient.

Besides the time problems, the replanning routine works well and is able to solve problems both when the plan fault is in the plan checking process

and when it is in the execution of a plan. Several different replanning problems have been tested; some of them were described in the previous section. Those replanning problems are of the kind where a plan is extended with an entire plan for another action, such as performing a **take off** command before performing a **fly** command. Examples where a subpart of a plan has been connected to a subpart of another plan have also been tested and worked as intended.

Nested dialogs and subdialogs have been tested and work as intended. The operator can jump back and forth between several ongoing dialogs, without problems. CEDERIC finds the most suiting ongoing dialog and reacts properly to the dialog exchange.

CEDERIC has been tested using both speech and text as a means of communication. When using speech, the phrase interpreted by the speech recognizer can be far from what the operator actually said. An example is when the phrase **Fly to the red hospital** is interpreted as **Watch the eight police cars**. This can of course be troublesome for the dialog manager which is trying to interpret the strange phrase. The experience from the tests is however that the result from the misinterpreted phrase is often faulty in some way, and CEDERIC will return an error message such as **I do not know what you mean with the phrase watch the eight police cars**. If the fault is minor, the similarity function may be able to recover from the error and interpret the utterance correctly. When using text, typing errors are common. They are often minor and the meaning of a sentence can easily be interpreted by the similarity function.

There is a greater risk that the operator uses grammatically incorrect sentences, leaves some words out, etc, when using text. When the meaning of the phrase is clear, even with some missing words, CEDERIC often interprets the phrase correctly, but when a major part of the sentence is missing, CEDERIC decides not to use the most similar case if it is not similar enough, and an error message is returned instead.

To sum up, there are some problems, mainly with the replanning routine, but in general, CEDERIC seems to work correctly and as intended for most of the tests.

## 8.3 User Tests

User tests are used for measuring the usability of a dialog system. The idea is to test how well the system works for an arbitrary user, to gain information about how a user interacts with the system, and to measure how easily a user can accomplish a task.

CEDERIC has been tested in user tests, together with the SGUI user interface and the Hazard simulator. The test persons are people that have no prior experiences of CEDERIC, but some of them have some experience of the WITAS RDE, which is similar to CEDERIC. A simple scenario including five missions and a model of the world where the helicopter is situated was given to each test person before the test began. The five missions used in the tests were:

1. Move the helicopter to an altitude of 20 meters. Make sure that the altitude really is 20 meters above the ground.
2. Move the helicopter to a position above, or close to the hospital.
3. Move the helicopter to a position above, or close to the school labeled with number 1 (the number can not be used to refer to the object in a dialog).
4. Make the dialog system play the video recorded ten minutes ago.
5. Move the helicopter to the base and make it land on the ground.

The test persons had no knowledge about which words or phrases to use when communicating with the system. The dialog between the user and the system was recorded for each mission and the turns were analyzed.

The first mission turned out to be hard. The ideal dialog is `ascend twenty meters` followed by the question `what is the altitude`. The problem is the word `ascend`, which has to be used for the system to recognize the first command. No other words with the same word class were included in the lexicon. Phrases such as `gain twenty meters`, `rise twenty meters` and longer phrases such as `fly to the altitude of twenty meters` occurred in the tests but could not be interpreted by the system. The

phrase `ascend to twenty meters` was recognized as `ascend twenty meters`, i.e. the system interpreted it as a relative altitude but the operator intended an absolute altitude. The operator recognized the error and could figure out the correct command. This is an example of when the system misinterprets a phrase, but it can be helpful for the operator anyway.

The ideal phrase for mission two is `fly to the hospital`, which some test persons used. Other phrases such as `fly to a position close to the hospital` occurred as well and was interpreted correctly.

In mission three, two schools were present in the world. The test persons did not know how to specify the choice of school, but the schools had different colors in the SGUI and the simulator. The thought behind this mission was to test how well the subdialog worked. To force a subdialog, the system did not have a plan for the phrase `fly to the red school`, only for the phrase `fly to the school` which resulted in a subdialog when there were several schools in the world. The ideal dialog is `fly to the school` which results in the reply `I have several schools to choose between, which one do you mean`. Then the operator answers `the red one` and the helicopter performs the action. Most test persons did follow the ideal dialog quite well. Instead of `fly to the school`, the similar phrase `go to the school` occurred and instead of `the red one`, the phrase `the red school` was used. These phrases were interpreted correctly and as the test person intended.

The formulation in mission four was not as leading as the test persons thought. The ideal phrase to show the video is `show me the video from ten minutes ago`. Variants such as `play video recorded ten minutes ago` and `show me the video you recorded ten minutes ago` were used successfully in the tests.

The last mission could easily be performed since by that time the test persons had learnt how to make the helicopter fly to a particular named position. The ideal dialog is `fly to the base` followed by a `land` command. Some test persons tried a compound phrase such as `please fly home to the base and land`. This was interpreted as `fly to the base` and the `land` command was not performed. However, the test person noticed this and could command the helicopter to land with the phrase `land please` when the helicopter had reported that it had completed flying to the base.

### 8.3.1 Results

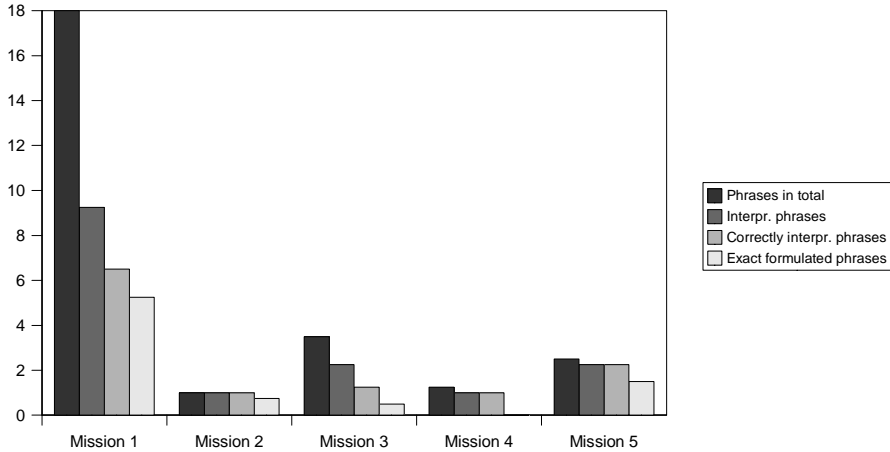


Figure 8.6: The result of the user test where the y-axis is the number of phrases on average per test person.

The dialogs and phrases for each test person and mission have been analyzed and categorized. Interesting results are:

- How many phrases a test person had to use on average to complete a mission,
- How many phrases the system could interpret on average for each mission,
- How many phrases that were interpreted correctly by the system on average for each mission,
- How many phrases that were uttered exactly as they were stored in the case base on average for each mission.

The result is given in Figure 8.6.

The result for mission 1 stands out because of the large number of phrases. As discussed in the previous section, this depends on the difficulty to find the correct word for ascending. There is also a larger gap between the number of interpreted phrases and the number of correctly interpreted phrases. This is however not as bad as it may look. Phrases such as `ascend to twenty meters` that was interpreted as `ascend twenty meters` and `make sure that the altitude is twenty meters` that was interpreted as `what is the altitude` were both wrongly interpreted but they did help the test person to perform the mission.

The results for the other four missions are rather similar to each other. Almost every phrase was interpreted correctly, even if it was not uttered exactly as it was stored in the case base. This indicates that the machine learning and similarity function were used and worked well. For mission 3 and mission 4, few phrases or no phrase at all was uttered exactly as it was stored in the case base, but despite this most of the communicated phrases were interpreted correctly.

The user test was not only a test of CEDERIC, but it also served as a test of how a user may communicate with the system to solve a problem. The test persons were not given any information before the test started how to communicate with the system, besides that he or she was free to use either spoken or written language. The results for mission 1 were a clear indication that the choice of wordings in the system disagreed with the test persons use of language. It would probably have been helpful if the system gave hints about how to communicate a command based on weak similarity. The phrase `gain twenty meters` could for example result in the answer `I do not know exactly what you mean by the phrase gain twenty meters; the most similar commands I know are ascend twenty meters and descend twenty meters if that is of any help. Please try again.`

# Chapter 9

## Conclusion

### 9.1 Retrospective

As CEDERIC is a dialog system built on a CBR foundation, it combines several different areas of research, and it is inspired by several mutually different systems. Chapter 4 contained descriptions of some systems that are similar to CEDERIC in some sense. The present section gives a comparison between them and CEDERIC and points out advantages and disadvantages with the systems.

#### 9.1.1 Predecessors within WITAS

The WITAS-Stanford Dialog System (described in section 4.1.2) and the WITAS RDE (described in chapter 5) are predecessors to CEDERIC, thus the systems have much in common. They are used in the same domain, with approximately the same interface to the actual UAV or simulator, and the dialogs implemented in the systems are rather similar. However, under the surface, the systems address different goals. CEDERIC focuses on the integration of machine learning into dialog systems to gain flexibility. The WITAS-Stanford Dialog System is also implemented with flexibility in focus, but does not include machine learning. The WITAS RDE uses a logic base that is partly inherited into CEDERIC. Both the WITAS-Stanford

Dialog System and, to a larger extent, the WITAS RDE implement multimodality, which is not yet implemented in CEDERIC.

### 9.1.2 Robot Dialog

The main purpose of CEDERIC is to control a robot, hence the focus of the dialog is on robot communication. Several other robotic dialog systems were presented in section 4.3. KAMRO, for example, is an advanced system including execution monitoring and explanation of error recovery. Features like these are very useful in a robotic dialog system, and particularly interesting for an application like CEDERIC. However, it is not yet implemented, but KAMRO serves as an inspiration for further implementations. Other systems such as Godot, Jijo-2 and Carl use some sort of learning such as socially embedded learning, similar to learning from explanation, which is implemented in CEDERIC. Unlike CEDERIC, they do not use machine learning to increase the dialog functionality.

### 9.1.3 Case-Based Reasoning and Planning

CEDERIC uses CBP for planning and replanning. Other systems using both CBP and other planning algorithms were described in section 4.4 and section 4.5. TRIPS is an interactive planning system that assists the user to construct effective plans for her work. It is a very interesting approach, and it would be interesting to implement interactive planning in CEDERIC, using CBR and CBP techniques. Today, CEDERIC uses planning only for solving problems without the interaction of a human user. SiN, described in section 4.5.1, is another system with planning capabilities. SiN also uses CBR as a problem solving method. However, SiN uses CCBR, which is a method where the dialog is an aid to gather more information to solve the problem, not the main feature of the system. SiN does not have a discourse model, and the dialog is very simple and strict. A step further is the Discourse Goal Stack Model presented in section 4.5.2. The model connects CCBR with a discourse model, which is very interesting. However, the machine learning method is only used as a problem solving technique, and the dialog does not improve over time, nor does it use the CCBR



technique to become better. The discourse model and the CCBR part are not integrated but are two separate parts of the system.

The most interesting approach, with respect to learning dialog systems, is the method described in section 4.5.3, by Murao et al. This dialog system uses CBR to learn new dialogs from an example corpus, similar to the way CEDERIC learns from dialog in its past experience. The cases are made up by an input from the user and the solution is a reply. Therefore a longer dialog is not captured in the case base in the same manner as CEDERIC does, using the distinction between plan cases and plan items. In combination with the lack of discourse information, its dialogs become rather simple and strict, compared to the dialogs in CEDERIC.

## 9.2 Main Results

CEDERIC is a dialog system implemented as an answer to the question: *How can an adaptable, learning, problem solving robotic dialog system be constructed so that it improves over time and incorporates advanced natural language features?*

that was asked in section 6.2.

This question was decomposed into six subquestions, which are recapitulated here together with a discussion:

*How can **machine learning** be provided in such a system?*

*How can **problem solving** be provided in such a system?*

As CEDERIC uses CBR, which is a machine learning method and a problem solving method, the two first questions can be answered together. CEDERIC uses CBR both as a machine learning technique and as a problem solving technique, which makes the basic system more coherent. It can find the best suiting case to apply when the operator does not utter a sentence exactly as it is specified in the system, and it can also save the plan case and use it again at a later time. It can also use a case for solving a problem where the words used belong to the same category, but are not identical to the word in the case base. The implementation of CEDERIC shows that a machine learning method as well as a problem solving method can be used successfully in a dialog system. In our case the method in both cases is CBR.

*How should the **discourse information** be represented?*

CEDERIC is equipped with a rather sophisticated discourse model that is well integrated into the system, giving it the ability to handle anaphoric references, subdialogs and topic management. These linguistic features make the dialog more natural and fluent.

*How can the system make **use of the knowledge to solve problems it has never seen before**?*

With the right prerequisites, CEDERIC can solve problems it has never seen before, and make qualified guesses about how to treat incomplete information by making effective use of the knowledge in the system. It is capable of finding and combining a subset of a case with a subset of another case to solve a new problem. A special case of this planning algorithm allows for preparing for the execution of a command the operator is asking for, by executing other necessary commands automatically. Hence CEDERIC is an example of how a system can make use of knowledge to solve problems it has never seen before.

*How can the system **learn from the user** if it can not solve a problem on its own?*

CEDERIC can learn from explanation and take directives from the operator, by recognizing a learning situation where the operator uses an unknown word, and execute a case that allows learning from explanation. The information is then saved in the system and can be reused later on.

*How can **special considerations regarding the physical robot, such as safety and graceful degradation, be taken care of**?*

When CEDERIC does not find a suitable case to be applied to a user input, the user is told so and may try to reformulate, hence the system uses some sort of graceful degradation where it can carry on with different tasks even if one input fails. However, in a system like CEDERIC, it is desirable to have a verbal system that explains what it has decided to do, before the plan is executed. This is particularly important in a machine learning and problem solving system, where the plan may be faulty or unsuitable. Such verbal behavior is not implemented in CEDERIC.

To return to the main question stated at the beginning of this section, has our work with CEDERIC identified a viable solution or approach for each of the stated subgoals? We think so, because CEDERIC is adaptable, learning and problem solving to a certain degree, although not comparable

to a human being. It improves over time, due to the increasing case base, and it can perform some advanced dialogs, but it is far from as intelligent as you and me. However, compared to other dialog systems, it can be considered intelligent in some sense.

Of course, much is in the eye of the beholder and a matter of interpretation of the question above, as is often the case in artificial intelligence. As shown above, CEDERIC does fulfill the requirements in the question and is an adaptable, learning, problem solving robotic dialog system that improves over time and incorporates advanced natural language features.

### 9.3 Future Work

CEDERIC has some weaknesses in the replanning routine, as described in section 7.5.5. A better implementation of the replanning routine is likely to result in a more robust and faster system. An integration of methods and ideas from an automatic planner into CEDERIC is an interesting and challenging task which would probably benefit the system. It is interesting to investigate how the planning routine may be implemented to be able to solve problems correctly and efficiently without losing the benefits from a learning system that tries different methods to solve a problem.

Despite its weaknesses, CEDERIC is flexible and mature enough to serve as a starting point for further implementations of various dialog features. One desirable feature is the ability for the system to explain a plan in natural language and get feedback from the operator. This can be used both for execution monitoring, where the operator accepts or rejects an automatically planned plan, and for interactive planning if a dialog about the plan occurs. Interactive planning is also interesting to implement in the planning phase, in order to obtain a usable dialog system in the robotic control domain. Approaches to interactive planning have been described, e.g., in the literature about TRIPS and in the automated planning literature.

The WITAS-Stanford Dialog System and the WITAS RDE both implement multimodality, i.e., a spoken input from the user is combined with a mouse gesture on a screen. The WITAS domain is particularly suited for multimodal input, where the user can point out areas on a map or indicate which vehicle she is talking about in a video stream. CEDERIC does not

support multimodality, but it would be interesting to investigate how it can be integrated to the system.

The similarity function in CEDERIC checks for equality or classification similarity, where two entities are classification similar if they belong to the same class in an ontology. However, classes may be related as well, and this relation may be taken into consideration in the similarity function. An investigation of an expansion of the ontology and the similarity function would be interesting. This includes issues such as how the information can best be represented in the ontology, and how new information can be included automatically by the system.

Earlier versions of CEDERIC saved aspects of the discourse model in the case base for each of the cases, and matched it against the current discourse situation when facing new cases. Due to lack of generality, this method was discarded in favor of saving the raw input from the whole dialog and using it instead. However, the thought of using the structured discourse is appealing and would be retried if a clean solution to the problem can be found.

# Bibliography

- [1] *Dynamic Memory: A Theory of Reminding and learning in computers and People*. Cambridge University Press, 1982.
- [2] *From Discourse to Logic*. Kluwer Academic Publishers, 1993.
- [3] *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley, 1994.
- [4] *Artificial Intelligence a Modern Approach*. Prentis-Hall, 1995.
- [5] *Features and Fluents: the Representation of Knowledge about Dynamic Systems*. Oxford University Press, 1995.
- [6] *Machine Learning*. McGraw-Hill, 1997.
- [7] *SmartKom: Foundations of Multimodal Dialogue Systems*. Springer Verlag, 2006.
- [8] Agnar Aamodt. Case-based reasoning; Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994.
- [9] David W. Aha, Leonard A. Breslow, and Hector Munoz-Avila. Conversational Case-Based Reasoning. *Applied Intelligence*, 14(1):9–32, 2001.

- 
- [10] Jan Alexanderson, Elisabeth Maier, and Norbert Reithinger. A Robust and Efficient Three-Layered Dialogue Component for Speech-to-Speech Translation System. Technical Report 50, Federal Ministry of Education, Science, 1994.
- [11] J. Allen, D. Byron, M. Dzikovska, G. Ferguson, and L. Galescu. Towards Conversational Human-Computer Interaction. *AI Magazine*, 22(4):27–37, 2001.
- [12] James Allen, George Ferguson, and Amanda Stent. An Architecture for More Realistic Conversational Systems. In *IUI '01: Proceedings of the 6th International Conference on Intelligent User Interfaces*, pages 1–8, 2001.
- [13] Peter J. Andersson. Hazard: a Framework Towards Connecting Artificial Intelligence and Robotics. In *IJCAI Workshop on Reasoning, Representation and Learning in Computer Games*, 2005.
- [14] Peter J. Andersson. Hazard: A Framework Towards Connecting Artificial Intelligence and Robotics. In *IJCAI Workshop on Representation, Reasoning and Learning in Computer Games*, 2005.
- [15] I. Androustopoulos, G.D. Ritchie, and P. Thanisch. Natural Language Interfaces to Databases – an Introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.
- [16] Hideki Asoh, Satoru Hayamizu, Isao Hara, Yoichi Motomura, Shotaro Akaho, and Toshihiro Matsui. Socially Embedded Learning of the Office-Conversant Mobile Robot Jijo-2. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 880–887, 1997.
- [17] S. Bennacef, L. Devillers, S. Rosset, and L. Lamel. Dialog in the RAIL-TEL Telephone-based System. In *Proceedings of ICSLP*, volume 1, pages 550–553, 1996.
- [18] Eric Bilange. A Task Independent Oral Dialogue Model. In *Proceedings of the Fifth Conference on European Chapter of the Association for Computational Linguistics*, pages 83–88, 1991.

- 
- [19] Karl Branting, James Lester, and Bradford Mott. Dialogue Management for Conversational Case-Based Reasoning. In *Proceedings of the Seventh European Conference on Case-Based Reasoning*, pages 77–90, 2004.
- [20] L. Breslow and D. Aha. NaCoDAE: Navy Conversational Decision Aids Environment. Technical Report AIC-97-018, NCARAI, Washington, DC, 1997.
- [21] Rolf Carlson. The Dialog Component in the Waxholm System. In *Proceedings of Twente Workshop on Language Technology. Dialogue Management in Natural Language Systems (TWLT 11)*, 1996.
- [22] Peter Clark and Bruce Porter. *KM - The Knowledge Machine 2.0: Users Manual*, 2004.
- [23] Ronald A. Cole, Joseph Mariani, Hans Uszkoreit, Annie Zaenen, and Victor Zue. Survey of the State of the Art in Human Language Technology. *Cambridge University Press*, 1997.
- [24] Patrick Doherty, Gösta Granlund, Krzysztof Kuchinski, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. The WITAS Unmanned Aerial Vehicle Project. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 747–755, 2000.
- [25] Patrick Doherty, Patrik Haslum, Fredrik Heintz, Torsten Merts, Tommy Persson, and Björn Wingman. A Distributed Architecture for Intelligent Unmanned Aerial Vehicle Experimentation. In *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems*, 2004.
- [26] Karolina Eliasson. An Integrated Discourse Model for a Case-Based Reasoning Dialogue System. In *SAIS-SSL event on Artificial Intelligence and Learning Systems*, 2005.
- [27] Karolina Eliasson. Integrating a Discourse Model with a Learning Case-Based Reasoning System. In *DIALOR-05: the 9th Workshop on the Semantics and Pragmatics of Dialogue*, 2005.

- 
- [28] Karolina Eliasson. Towards a Robotic Dialogue System with Learning and Planning Capabilities. In *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2005.
- [29] George Ferguson and James F. Allen. TRIPS: An Integrated Intelligent Problem-Solving Assistant. In *AAAI '98/IAAI '98: Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 567–572, 1998.
- [30] R.E Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3–4):182–208, 1971.
- [31] T. Finin, R. Fritzon, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, 1994.
- [32] Matthew Frampton and Oliver Lemon. Reinforcement Learning of Dialogue Strategies Using the Users's Last Dialogue Act. In *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2005.
- [33] Jeroen Groenendijk and Martin Stokhof. On the Semantics of Questions and the Pragmatics of Answers. In *Varieties of Formal Semantics*, pages 143–170, 1984.
- [34] Nobuo Inui, Toshiaki Ebe, Bipin Indurkha, and Yashiyuki Kotani. A Case-Based Natural Language Dialogue System Using Dialogue Act. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 193–198, 2001.
- [35] T. Laengle, T. C. Lueth, G. Herzog, E. Stopp, and G. Kamstrup. KANTRA – A Natural Language Interface for Intelligent Robots. *Intelligent Autonomous Systems*, pages 365–372, 1995.
- [36] Lynn Lambert and Sandra Carberry. A Tripartite Plan-Based Model of Dialogue. In *Proceedings of the 29th Annual Meeting on Association for Computational Linguistics*, pages 47–54, 1991.




- 
- [37] Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. Information States in a Multi-modal Dialogue System for Human-Robot Conversation. In *5th Workshop on Formal Semantics and Pragmatics of Dialogue*, 2001.
- [38] Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. The WITAS Multi-Modal Dialogue System. In *Proceedings of EuroSpeech*, pages 1559–1562, 2001.
- [39] Oliver Lemon, Alexander Gruenstein, and Stanley Peters. Collaborative Activities and Multi-tasking in Dialogue Systems. *Traitement Automatique des Langues (TAL), special issue in dialogue*, 43(2):131–154, 2002.
- [40] Ester Levin, Roberto Pieraccini, and Wieland Eckert. A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies. *Journal of Artificial Intelligence Research*, 8(1):105–133, 2000.
- [41] L. Seabra Lopes. Carl: from Situated Activity to Language Level Interaction and Learning. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, pages 890–896, 2002.
- [42] David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The Open Agent Architecture: A Framework For Building Distributed Software Systems. *Applied Artificial Intelligence*, 13(1-2):91–128, 1999.
- [43] Hector Munoz-Avila, Ralph Bergmann Manuela Veluso, and Erica Melis. Case-based Reasoning Applied to Planning Tasks. *Case-Based Reasoning Technology: From Foundations to Applications*, pages 169–199, 1998.
- [44] Héctor Muñoz-Avila, David W. Aha, Dana S. Nau, Rosina Weber, Len Breslow, and Fusun Yaman. SiN: Integrating Case-based Reasoning with Task Decomposition. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 999–1004, 2001.
- [45] Hiroya Murao, Nobuo Kawaguchi, Shigeki Matsubara, and Yasuyoshi Inagaki. Example-based Query Generation for Spontaneous Speech.

- IEICE Transactions on Information and Systems*, pages 357–364, 2005.
- [46] Hiroya Murao, Nobuo Kawaguchi, Shigeki Matsubara, Yukiko Yamaguchi, and Yasuyoshi Inagaki. Example-based Spoken Dialogue System using WOZ System Log. In *SIGdial Workshop on Discourse and Dialogue*, pages 140–148, 2003.
- [47] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 968–973, 1999.
- [48] Bernhard Nebel and Jana Koehler. Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis. *Artificial Intelligence*, 76(1-2):427–454, 1995.
- [49] Nils J. Nilsson. Shakey the Robot. Technical Report 323, AI Center, SRI International, 1984.
- [50] K. Nordberg, P. Doherty, P-E. Forssen, J. Wiklund, and P. Andersson. A Flexible Runtime System for Image Processing in a Distributed Computational Environment for an Unmanned Aerial Vehicle. *special issues of the International Journal of Pattern Recognition and Artificial Intelligence*, 2003.
- [51] Per-Olof Petterson and Patrick Doherty. Probabilistic Roadmap Based Path Planning for Autonomous Unmanned Aerial Vehicles. In *ICAPS Workshop on Connecting Planning and Theory with Practice*, 2004.
- [52] Norbert Pfeleger, Jan Alexandersson, and Tilman Becker. A Robust and Generic Discourse Model for Multimodal Dialogue. In *Workshop Notes of the IJCAI-03 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, 2003.
- [53] A. A. Sanderman, J. Sturm, E. A. den Os, L. Boves, and A. H. M. Cremers. Evaluation of the Dutch Train Timetable Information System developed in the Arise project. In *Proceedings of the 4th IEEE workshop on Interactive Voice Technology for Telecommunications Applications*, pages 91–96, 1998.

- 
- [54] Erik Sandewall. Cognitive Robotics Logic and its Metatheory: Features and Fluents Revisited. *Linköping Electronic Articles in Computer and Information Science*, 3, 1998.
- [55] Erik Sandewall, Patrick Doherty, Oliver Lemon, and Stanley Peters. Words at the Right Time: Real-Time Dialogues with the WITAS Unmanned Aerial Vehicle. In *Proceedings of the 26th Annual German Conference in AI*, pages 52–63, 2003.
- [56] Erik Sandewall, Hannes Lindblom, and Björn Husberg. Integration of Live Video in a System for Natural Language Dialog with a Robot. In *DIALOR-05: the 9th Workshop on the Semantics and Pragmatics of Dialogue*, 2005.
- [57] Satoko Shiga and Seishi Okamoto. Case-based Natural Language Dialogue System using Facial Expressions. In *CATALOG-04: the 8th workshop on the semantics and pragmatics of dialogue*, 2004.
- [58] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research*, 16(1):105–133, 2002.
- [59] Luca Spalazzi. A Survey on Case-Based Planning. *Artificial Intelligence Review*, 16(1):3–36, 2001.
- [60] Christian Theobalt, Johan Bos, Tim Chapman, Arturo Espinosa-Romero, Mark Fraser, Gillian Hayes, Ewan Klein, Tetsushi Oka, and Richard Reeve. Talking to Godot: Dialogue with a Mobile Robot. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1338–1343, 2002.
- [61] Alan Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, 1950.
- [62] Steve Young. Talking to Machines (Statistically Speaking). In *Proceedings of the International Conference on Spoken Language Processing*, 2002.



 <b>Linköpings universitet</b>	<b>Avdelning, Institution</b> Division, Department  <b>AIICS,</b> Dept. of Computer and Information Science 581 83 LINKÖPING	<b>Datum</b> Date  May 8, 2006
<b>Språk</b> Language  <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English  <input type="checkbox"/> _____	<b>Rapporttyp</b> Report category  <input checked="" type="checkbox"/> Licentiatavhandling <input type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	<b>ISBN</b> 91-85523-78-X <hr/> <b>ISRN</b> LiU-Tek-Lic-2006:29 <hr/> <b>Serietitel och serienummer ISSN</b> Title of series, numbering <u>0280-7971</u> <hr/> Linköping Studies in Science and Technology Thesis No. 1248
<b>URL för elektronisk version</b> <a href="http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-6402">http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-6402</a>		
<b>Titel</b> Title  The Use of Case-Based Reasoning in a Human-Robot Dialog System  <b>Författare</b> Author Karolina Eliasson		
<b>Sammanfattning</b> Abstract  <p>As long as there have been computers, one goal has been to be able to communicate with them using natural language. It has turned out to be very hard to implement a dialog system that performs as well as a human being in an unrestricted domain, hence most dialog systems today work in small, restricted domains where the permitted dialog is fully controlled by the system.</p> <p>In this thesis we present two dialog systems for communicating with an autonomous agent:</p> <p>The first system, the WITAS RDE, focuses on constructing a simple and failsafe dialog system including a graphical user interface with multimodality features, a dialog manager, a simulator, and development infrastructures that provides the services that are needed for the development, demonstration, and validation of the dialog system. The system has been tested during an actual flight connected to an unmanned aerial vehicle.</p> <p>The second system, CEDERIC, is a successor of the dialog manager in the WITAS RDE. It is equipped with a built-in machine learning algorithm to be able to learn new phrases and dialogs over time using past experiences, hence the dialog is not necessarily fully controlled by the system. It also includes a discourse model to be able to keep track of the dialog history and topics, to resolve references and maintain subdialogs. CEDERIC has been evaluated through simulation tests and user tests with good results.</p>		
<b>Nyckelord</b> Keywords Dialog Manager, Machine Learning, Case-Based Reasoning, Case-Based Planning, Helicopter, Natural Language		