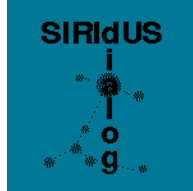

Dialogue Moves in Natural Command Languages

J. Gabriel Amores and José F. Quesada

Distribution: Public



Specification, Interaction and Reconfiguration in
Dialogue Understanding Systems: IST-1999-10516

Deliverable D1.1

September, 2000

Specification, Interaction and Reconfiguration in Dialogue Understanding
Systems: IST-1999-10516

Göteborg University

Department of Linguistics

SRI Cambridge

Natural Language Processing Group

Telefónica Investigación y Desarrollo SA Unipersonal

Speech Technology Division

Universität des Saarlandes

Department of Computational Linguistics

Universidad de Sevilla

Julietta Research Group in Natural Language Processing

For copies of reports, updates on project activities and other SIRIDUS-related information, contact:

The SIRIDUS Project Administrator
SRI International
23 Millers Yard,
Mill Lane,
Cambridge, United Kingdom
CB2 1RQ
milward@cam.sri.com

See also our internet homepage <http://www.cam.sri.com/siridus>

©2000, The Individual Authors

No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.

Contents

1	Introduction	8
2	Information Seeking, Natural Command Language and Negotiative Dialogues	12
2.1	Some initial examples	13
2.1.1	Dialogue 2.1: GoDiS	13
2.1.2	Dialogue 2.2: MIDAS	13
2.1.3	Dialogue 2.3: TeleDelfos	14
2.1.4	Dialogue 2.4: Natural Language Unix Interface	15
2.1.5	Dialogue 2.5: Washing Machine	15
2.1.6	Dialogue 2.6: Travel Machine	15
2.2	Discriminatory features	16
2.2.1	Human-Machine Interaction	16
2.2.2	Flow of Information	16
2.2.3	Underlying task	16

2.3	An initial dialogue classification	17
3	Natural Command Language Dialogues	19
3.1	Command Languages	20
3.1.1	Command Languages from the Dialogue Perspective . . .	20
3.1.2	Computational Characteristics of Command Languages . .	21
3.1.3	A Command Language sample	22
3.2	Command Language Systems	23
3.3	Natural Command Languages	23
3.3.1	Computational Properties of Natural Command Languages	25
3.4	Natural Command Language Systems	26
3.5	Natural Command Language Dialogues	26
3.5.1	Dialogue Commands	28
3.6	Natural Command Language Dialogue Systems	29
3.7	A Natural Language Interface for Virtual Reality Systems	29
4	Dialogue Moves in Natural Command Language Dialogues	34
4.1	Informational Components	35
4.2	Dialogue Moves	36
4.2.1	askCommand	37
4.2.2	specifyCommand	38

4.2.3	informExecution	38
4.2.4	askParameter	40
4.2.5	specifyParameter	41
4.2.6	askConfirmation	41
4.2.7	answerYN	42
4.2.8	askContinuation	42
4.2.9	askRepeat	43
4.2.10	askHelp	44
4.2.11	answerHelp	45
4.2.12	errorRecovery	45
4.2.13	greet	46
4.2.14	quit	46
4.3	Annotations with NCL Dialogue Moves	46
4.3.1	Dialogue Moves in Dialogue 2.1: GoDiS	47
4.3.2	Dialogue Moves in Dialogue 2.2: MIDAS	47
4.3.3	Dialogue Moves in Dialogue 2.3: TeleDelfos	48
4.3.4	Dialogue Moves in Dialogue 2.4: Natural Language Unix Interface	49
4.4	Control Strategy	49
5	Comparing Natural Command Language Moves with Other Move Cod- ing Schemes	53

5.1	Scheme Layers	53
5.2	Move Taxonomies	54
5.3	A Comparison between DRI, TRAINS and NCL Moves	56
6	Natural Command Language Dialogue Moves for the Spanish Telephone Command Dialogue Corpus	60
6.1	Annotating the Spanish Telephone Command Dialogue Corpus . .	60

Chapter 1

Introduction

One of the main goals of the Siridus project, as explicitly stated in the general description of the work plan, is to extend the range of types of dialogue to which the Information State Update (ISU) view of dialogue is applicable.

The ISU approach to the study, design and implementation of dialogue systems is one of the main contributions of the EU 4th Framework project TRINDI (Task Oriented Instructional Dialogue) [Traum *et al*, 1999]. This project has promoted a theoretical account of dialogue in which conversational acts are viewed as objects which update, revise and align the Informational States of the participants in the dialogue.

Roughly speaking, the term *information state* means the information stored internally by an agent, in this case a dialogue system. More precisely, the information state of a dialogue includes all the cumulative additions from previous actions in the dialogue, which, taking into account the structure and goals of the dialogue, motivates and determines the future actions in the dialogue.

By its own nature and functionality, the information state of a dialogue is a dynamic structure, that evolves (is updated) from the observation and performance of *dialogue moves*.

Accordingly, from an operational perspective, a *dialogue move engine* can be described as a module in charge of two main functions: first, updating the information

state on the basis of observed dialogue moves, and second, selecting appropriate moves to be performed.

This approach has been applied to the study of *information seeking dialogues* within the scope of TRINDI. In these dialogues, the user seeks information from a system which knows well the overall task of the user (hence *task oriented*) but which does not know the particular parameters that the user is interested in. In general, the user does not know which parameters the system requires and so, if the parameters are not supplied by the user, then the system needs to query the user for them.

In the SIRIDUS project, two new types of dialogues will be considered:

- *Natural Command Language* dialogues. In this case, the user does know precisely what procedure (in the system's terms) he wants the system to execute and he will generally attempt to give the system all this information at once. The system does not know which task the user wishes to carry out, although there will generally be only a finite and fairly small number of them. Consequently, these dialogues may be of quite a different character from information seeking dialogues. The system needs to identify the task and the parameters from the user's offering, in what the user will expect to be almost a *one-shot* environment. Follow-up dialogues will be expected to function mostly as disambiguation exercises with respect to the particular task domain of the system. Prime examples of such natural command language dialogues include speech interfaces to personal service devices such as video-recorders, cookers, telephones and personal computers.
- *Negotiative* dialogues. Here, the user and the system *jointly* attempt to determine a piece of information or achieve a goal. Such dialogues must be mixed initiative dialogues and hence differ from the simpler cases where the system quizzes a user who is thereby treated as essentially an answer-supplier.

This document concentrates on the study of Natural Command Language dialogues from the Information State Update perspective, and on the specification of Dialogue Moves for this type of dialogues.

Chapter 2 offers a description of the discriminating features of these three types of dialogue from the analysis of a set of initial examples. Three basic features will

be proposed, whose values (for each type of dialogue) are summarised in the table below:

<i>Dialogue Types</i>	<i>Key words</i>	<i>Human-Machine Interaction</i>	<i>Flow of information</i>	<i>Underlying task</i>
Information Seeking Dialogues	Consult	Repository of knowledge	Shared knowledge	Information seeking task
Natural Command Language Dialogues	Detect, Complete and Execute	Operational system	Shared knowledge & Change of the state of the world	Command execution task
Negotiative Dialogues	Propose, Suggest, Negotiate and Agree	Goal negotiation system	System-user convergence	Goal agreement task

Next, chapter 3 describes the basic functional and operational relations between the elements involved in a dialogue management system based on Natural Command Languages. Our study distinguishes two levels: an interaction specification level (languages or dialogue models) and an execution and management level (system level). On the other hand, different communication models are also differentiated, distinguishing Command Languages (CL), Natural Command Languages (NCL) and Natural Command Language Dialogues (NCLD). The resulting functional structure obtained from this model is graphically represented by the figure 1.1. Also, chapter 3 describes a project aimed at the use of Natural Command Languages in a virtual reality environment.

Chapter 4 concentrates on the study of Dialogue Moves for Natural Command Languages, using the Information State Theory as a conceptual basis. Therefore, a preliminary analysis of the features of the informational components, dialogue moves and control strategy will be provided.

As regards Dialogue Moves, an initial general taxonomy will be described which may be viewed as an extension to Information Seeking dialogues. This chapter includes annotated dialogue examples taken from different environments, using

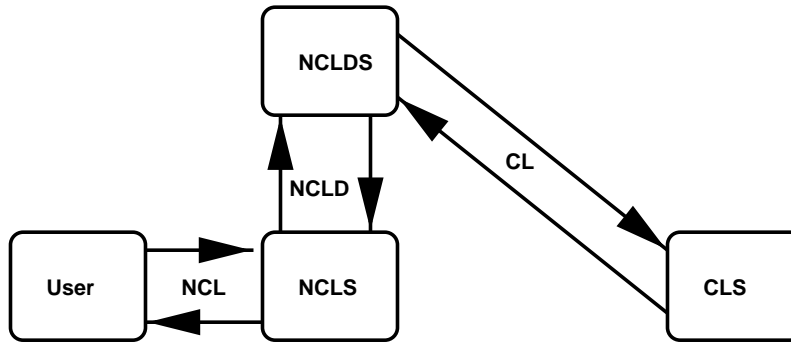


Figure 1.1: A functional interaction model for NCLD

our proposed dialogue moves.

Chapter 5 compares the proposed taxonomy of Dialogue Moves for Natural Command Languages with other Move Coding Schemes.

Finally, chapter 6 shows an annotated real dialogue taken from the Automatic Telephone Operator Scenario corpus (see *Deliverable 3.1: User Requirements on a Natural Command Language Dialogue System*).

Chapter 2

Information Seeking, Natural Command Language and Negotiative Dialogues

In order to specify the main properties and characteristics of *Natural Command Language* dialogues, our first goal in this chapter concentrates on the description of the three dialogue types proposed.

Section 1 presents a representative small set of sample dialogues, which will allow us to illustrate the main differences among them. Specifically, we propose three main features to differentiate and classify the dialogues: first, the human-machine interaction strategy related to the perception of the system from the user; second, the pattern of the flow of information, which implies the progress of the information state during the interaction, and, third, the underlying tasks.

Section 2 is an initial and informal introduction to the study of the different values that the previous three features take in different types of dialogue.

This discussion allows for a classification of the three types of dialogue. Section 3 presents this first classification and applies the ideas to the dialogue samples presented in section 1.

2.1 Some initial examples

Consider the following set of dialogue extracts:¹

2.1.1 Dialogue 2.1: GoDiS

- **S(1)**: Welcome to the travel agency!
- **U(1)**: flights to paris
- **S(2)**: What city do you want to go from?
- **U(2)**: london
- **S(3)**: Do you want a return ticket?
- **U(3)**: yes, as cheap as possible
- **S(4)**: What month do you want to leave?
- **U(4)**: april, please
- **S(5)**: It will cost 2933 crowns
- **U(5)**: thanks, bye!
- **S(6)**: Thank you for your visit!

2.1.2 Dialogue 2.2: MIDAS

- **S(1)**: do you want a route?
- **U(1)**: I travel to London.
- **S(2)**: well well well, do you want a route?
- **U(2)**: Yes.

¹The examples have been artificially built in order to emphasize their differences as well as the main points from the dialogue system specification, design and implementation perspectives. In these dialogues, we are supposing a Human-Machine interface where **S** stands for system, and **U** stands for user.

- **S(3)**: when do you go?
- **U(3)**: I will start at nine am.
- **S(4)**: where do you start?
- **U(4)**: I like to start in Cambridge.
- **S(5)**: Please wait while I am calculating your route ...
- **S(6)**: Your route from cambridge to london

From:	Cambridge, Gloucs
To:	London, GtLon
Total Distance:	111.7 Miles
Total Time:	2 hr 23 min

- **S(7)**: bye bye!

2.1.3 Dialogue 2.3: TeleDelfos

- **S(1)**: Welcome to Delfos. Specify a function, please.
- **U(1)**: good morning, I would like to place a call.
- **S(2)**: Yes, who would you like to call?
- **U(2)**: no, I want to call 1 2 3 4 collect.
- **S(3)**: Do you really want to call 1 2 3 4 collect?
- **U(3)**: no, 2 1 3 4.
- **S(4)**: Do you really want to call 2 1 3 4 collect?
- **U(4)**: yes, please.
- **S(5)**: CALL COLLECT EXECUTION
- **S(6)**: Do you want to continue?
- **U(5)**: no thanks, goodbye.
- **S(7)**: Thanks for using the TeleDelfos system. Goodbye.

2.1.4 Dialogue 2.4: Natural Language Unix Interface

- **S(1):** Welcome to NL-Unix Interface. Specify a command, please.
- **U(1):** Please list the content of the test directory.
- **S(2):** Do you want the directory list ordered by name?
- **U(2):** Yes, please.
- **S(3):** DIRECTORY LIST. Done. What next?
- **U(3):** Delete the file sample.
- **S(4):** Do you want to delete the file sample?
- **U(4):** yes, exactly.
- **S(5):** I've deleted the file sample. Do you want to continue?
- **U(5):** no, that's all.
- **S(6):** Closing the NL-Unix Interface.

2.1.5 Dialogue 2.5: Washing Machine

- **U(1):** Low temperature whites wash please.
- **S(1):** My low temperature cycle is suitable for all loads. For whites only, I have a high temperature cycle and an economy cycle.
- **U(2):** Economy please.
- **S(2):** Please say "Start program" when you want the economy program to begin.
- **U(3):** Please start program
- **S(3):** Economy program has just begun.

2.1.6 Dialogue 2.6: Travel Machine

- **S(1):** What time do you want to go?
- **U(1):** What time do the trains leave?

2.2 Discriminatory features

An initial and informal analysis of these dialogues will show important differences from the following points of view:

2.2.1 Human-Machine Interaction

Which involves the perception of the system by the user. At this level, we can differentiate three main interaction models:

1. Repository of knowledge
2. Operational system
3. Goal negotiation system

2.2.2 Flow of Information

This item includes the amount of information that is required to represent the state of the dialogue, the *real* (state of the world) consequences of the dialogue itself, and the mutual relationships between the internal and shared information of both human and machine. Again, we will distinguish three main models at this level:

1. Increased amounts of shared knowledge between user and system, but with no effect on the world
2. Increased amounts of shared knowledge between user and system, and the state of the world may change because the system is undertaking actions.
3. Increased convergence between user and system goals and possible change of the state of the world.

2.2.3 Underlying task

Where we differentiate the following three tasks:

1. Information seeking task: an action to obtain from a system of stored information, a particular form of the information in order to facilitate planning or confirmation that an action has been accomplished.
2. Command execution task: an action to determine which command (from the set of available commands in the system) the user wants to perform, and the specific parameters for the command, with special care about confirmation since the execution of the command will change something in the world.
3. Goal agreement task: is one in which the user has a task to perform, but does not necessarily know how to go about it. It also could be the case that the system cannot accomplish the task. Then the system has to negotiate the accomplishment of the goal if possible, giving the user choices as they become available.

2.3 An initial dialogue classification

The table below sketches the main characteristics of the three types of dialogues considered:

<i>Dialogue Types</i>	<i>Key words</i>	<i>Human-Machine Interaction</i>	<i>Flow of information</i>	<i>Underlying task</i>
Information Seeking Dialogues	Consult	Repository of knowledge	Shared knowledge	Information seeking task
Natural Command Language Dialogues	Detect, Complete and Execute	Operational system	Shared knowledge & Change of the state of the world	Command execution task
Negotiative Dialogues	Propose, Suggest, Negotiate and Agree	Goal negotiation system	System-user convergence	Goal agreement task

According to this classification, dialogues 1 and 2 of section 1 belong to the *Information Seeking* type. In some sense, these dialogues try to *consult* (obtain) some

information from the *repository of knowledge* of the system (in the first dialogue, a travel fare database, and in the second, an autoroute database). In order to accomplish this (*information seeking task*) the dialogue system asks the user several questions to fix the open parameters of the consultation. As a result, at the end both the system and the user *share* the specific task and parameters. Besides, if the dialogue has been successful, the user will share the specific knowledge they wanted from the system.

Dialogues 3 and 4 represent systems where the dialogue is the user interface of a module which is able to *execute* several actions. So, after reading the dialogues we can reasonably think that under the dialogue system there is a lower level module that offers the execution of a set of actions. These actions must be activated by a *command language*. Therefore, the focus of the dialogue system is a *command execution task*, and the dialogue system helps the user in the selection of the specific command, the completion of the mandatory parameters of the command, and the overall control of the dialogue (which includes implicit and explicit confirmation, dialogue and task continuation, etc). The information flow pattern is similar to that of information seeking dialogues, although this time the dialogue system, through its connection with the low level execution module will be able to send the command to this module (once confirmed) and this probably will change the *state of the world*.

Finally, dialogues 5 and 6 show the properties of *Negotiative* dialogues. In these cases, the user doesn't have a detailed and specific goal, or even if he/she does have a goal, he/she doesn't have a concrete plan to achieve, so both user and system have to *propose, suggest, negotiate and agree* the goal and the plan. Therefore, interactions are oriented towards a *system-user convergence*, and the system's main task will be the *goal agreement* with the user.

Chapter 3

Natural Command Language Dialogues

Briefly and precisely speaking, a *Natural Command Language* is a command language expressed through the medium of natural language.

Consequently, a *Natural Command Language Dialogue* is a dialogue (usually between a human and a computer) using a natural command language.

Besides, the dialogue level may incorporate a new (emergent) functionality as a set of commands over the original low level command language. For instance, a dialogue management module for a telephone system may incorporate the command *retry last call*, although this is not a proper command of the telephone command language (it is not supported by the functional module in charge of the execution of the command language).

This chapter will present these ideas and the related notions. First, section 1 begins with the notion of Command Language (CL). Next sections present the notions of Command Language System (CLS), Natural Command Language (NCL), Natural Command Language System (NCLS), Natural Command Language Dialogue (NCLD) and Natural Command Language Dialogue System (NCLDS).

This set of notions are the result of a two-dimensional classification. On the one hand, we distinguish two levels: an interaction specification level (languages or

dialogue models) and an execution and management level (system level). On the other hand, different communication models are also differentiated, distinguishing Command Languages (CL), Natural Command Languages (NCL) and Natural Command Language Dialogues (NCLD).

		Interaction specification	Execution and management
Communication models	Command language	CL	CLS
	Naturally expressed	NCL	NCLS
	Through a dialogue	NCLD	NCLDS

Finally, section 7 illustrates these ideas and the corresponding functional architecture with an example: A Natural Language Interface for Virtual Reality Systems.

3.1 Command Languages

3.1.1 Command Languages from the Dialogue Perspective

Command languages are artificially constructed languages designed for interacting with an operating system through input of typed text. They are primarily designed for interactive use and thereby support very simple short *dialogues* consisting of a User Command followed by a System Response.

User Commands typically consist of one major action denoting term plus any number of optional and mandatory flags, option names and values.

System Responses provide feedback in the way of acknowledgments (“ok”, “done”) or error messages (“could not find file”, “permission denied”) or they may be empty.

Languages are also designed to support programmability. Complex commands may be defined out of simple ones using simple sequencing (do A; do B . . .),

sequencing plus context dependence (`do A; if p do B else do C`), function definition and application (`define fn X: A -> B; apply X to a`) and so forth. However, expressions of highly complex programs through Command Languages may be unwieldy, inefficient or simply impossible.

Not all operating system functionality may be accessible through Command Languages.

HCI research indicates that command languages are generally held to be flexible (compared to menu-based interaction), capable of supporting user-initiative and convenient for simple programming.

Design principles for good command languages include making them precise, compact, easy to write and read, easy to learn, simple, easy to remember, expressive, and consistent (e.g. in argument ordering).

3.1.2 Computational Characteristics of Command Languages

Generally, a command language is designed as a user interface to help with the access and control operations of the system. This way, the basic design goals for command languages are:

- Precision (in order to avoid ambiguity in the relation between the user and the system)
- Compactness
- Simplicity

Other desirable goals of language design that are important from the point of view of the usability of the language are:

- Ease in writing and reading
- Speed in learning
- Ease of retention over time

Finally, a third group of higher level goals from the point of view of the possible use of the command language in a dialogue system environment are:

- Close correspondance between reality and the notation
- Convenience in carrying out manipulations relevant to the users' tasks
- Flexibility to accommodate novice and expert users
- Expressiveness to encourage creativity

3.1.3 A Command Language sample

Unix is perhaps one of the best well-known command languages. This language exhibits several important properties:

1. As an operating system, the key grammatical structure of Unix is that of *command*. Besides, there is a formally specified grammar for the language, as well as a well-defined semantics.
2. The language is *compositional*, which is a good base for its programmability. The high level capabilities of different shells extend this feature, so that, in some sense, Unix may be considered as a simple programming language.
3. It is *execution-oriented*. By this notion, we refer to the fact that this language defines the linguistic (lexical -tokenization-, grammatical and semantic rules) that allow the user to execute actions in a system and to obtain the results.

As a language, the basic notion is that of sentence. The notion of dialogue is beyond a command language. Unix, as a language, defines grammatical and ungrammatical utterances (commands, here). Even, it may define the semantics associated to grammatical utterances (the notion of class' contract used in Java illustrates this idea). But a long-term interaction (a dialogue) is beyond the nature and functionality of a Command Language.

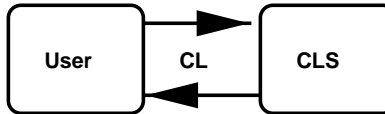


Figure 3.1: User (CL) CLS

and Language Systems

Language is a language in the strictest sense of the Formal Language. To make it operational in a computational environment we will implement a *Command Language System* for this command language. It will be able to receive input in the command language, process it (at syntactic and semantic levels), determine its correctness and execute it.

The Command Language System as the interpreter or compiler for the Command Language.

A specific version of Linux (let's call it XLinux) may be considered a Command Language System for the Command Language Unix.

The interaction between a User and a Command Language System (such as XLinux) for the Command Language (such as Unix) is represented in Figure 3.1.

Command Languages

A *Natural Command Language* is a command language expressed through the natural language. A Natural Command Language will at least express the meaning of the natural language, and will satisfy the following properties:

1. It will use the natural language vocabulary as far as possible.

2. It will use natural subcategorization frames (e.g. optional elements to be excluded, optional PPs, required items to be subcategorized for), especially

as far as semantic selectional restrictions is concerned. In this sense, an argument semantic type might determine the precise meaning of the function.

3. They reflect natural syntactic order (e.g. in English Verb-Obj-Obj for imperatives - not all commands need be imperative though). However, from a structural point of view, NCLs tend to reflect patterns of spoken language, rather than written language. They may, for example, make use of topicalised patterns (“*It is John I want to call, not Jim*”); permit some use of context dependent expressions such as pronouns (“*that one*”, “*it*”, and so forth), ellipsis (“*Send the account to Jim. And to Mary*”), etc.
4. Constitute a sub-language, with reasonably clear boundaries. It is important that users can learn the degree of expressivity of the language, use elements of it compositionally and yet not expect any arbitrary NL expression to be machine understandable.
5. New functionality in the system ought to be able to be accommodated within the language without altering language structural properties. Users ought to be able to introduce to some degree their own modes of expression and have the system understand them.
6. Be domain-independent. Ideally, issuing commands in a new domain should not necessarily mean learning a completely new language. Lexical items might vary (and extra ambiguity might be introduced) but again structural principles ought not to vary. Taking this idea to an extreme would basically reduce the NCL grammar to just one production:

Function -> FunctionWord Parameters*.

However, this desideratum may conflict somewhat with the previous goal (4) “Naturally occurring sublanguages” may very well have domain dependent restrictions on syntax and semantics (e.g. in domain X always interpret K as P), even deviant rules, and may lead to inefficiency from a computational point of view.

7. Cover system outputs, as well as inputs. If system outputs can be expressed in the sublanguage, then this may encourage the user to use it and help him to learn it. Feedback messages will need to be sensitive to problems in the linguistic expression of commands; for example, potential ambiguities.

In conclusion, we take NCLs as the set of input and output natural language expressions which are acceptable in a given application domain. This domain is semantically defined by the functions (commands) known by the user and the system, and the natural language vocabulary which may be used to express those commands. In addition, it should contain metalinguistic patterns and expressions typical of human-like interaction.

3.3.1 Computational Properties of Natural Command Languages

Basically, a Natural Command Language will then be a *naturally*-extended Command Language.

For instance, the following Unix sentence:

```
find /home *test* | wc
```

should be *naturally*-expressed as:

Please, tell me how many files whose name contains the expression test are contained in the /home directory

But let's consider this classical goal of the programming languages field in relation with command languages.

A priori, the use of a natural language as the means of expression poses a problem if we wish to maintain precision, compactness and simplicity.

On the other hand, the rest of goals will take advantage of the use of a natural language: ease in writing and reading, speed in learning, ease of retention over time, close correspondance between reality and the notation, convenience in carrying out manipulations relevant to the users' tasks, flexibility to accommodate novice and expert users and expressiveness to encourage creativity.

The question that appears next is whether it is reasonable to make a compromise between precision and ease to use. Traditionally, the computer-oriented community has rejected the use of natural language interfaces to computer systems in order to

avoid the ambiguity characteristic of natural languages. But we think it is possible to use a Natural Command Language approach if the domain is suitably restricted.

On the other hand, natural language output is a reasonable goal.

In summary, natural command languages are command languages expressed using restricted natural (sub-)languages. We should expect to use the vocabulary and the grammatical structure of a natural language, but restricted to the domain of the command language (which implies automatic disambiguation strategies).

3.4 Natural Command Language Systems

Again, to be computationally operational, a Natural Command Language will need a system able to understand (recognise, parse and extract the meaning of) the utterances in that NCL.

Keeping the comparison, a Natural Command Language System will be the interpreter or compiler of a Natural Command Language.

Figure 3.2 represents the two new elements once incorporated to the relationship framework defined by the user and the Command Language System. It is worth noting that Natural Command Language and Natural Command Language Systems define an interface (a natural language interface) between the user and the system that actually executes the commands. Thus, from a functional point of view, a Natural Command Language System may be seen as a translation tool where the Natural Command Language is the source language and the Command Language is the target language.

Also, this functional picture emphasizes a uniform approach for Natural Command Language analysis and generation.

3.5 Natural Command Language Dialogues

Natural Command Language Dialogues (NCLD) are artificially constructed models (including knowledge representation and reasoning) able to guide the interac-



Figure 3.2: User (NCL) NCLS (CL) CLS

between the different parts involved in a dialogue based on a NCL.

D will at least allow the following kind of phenomena:

Multiple Commands NCL

NCLD must be able to manage different commands and dialogue operations. So, one of the main functions of NCLD models will be Command Detection.

Context Dependency

Only at the dialogue level is it possible to understand anaphora, ellipsis and other context dependent constructions.

From the dialogue system design perspective, the treatment of these discourse phenomena will imply the representation and storage of the whole dialogue history so far.

Man-Machine Interaction

Naturalness and flexibility of the flow of interactions (restricted to the linguistic limitations of the underlying NCL), relevance and adequacy of the outputs, consistency (order of arguments in commands), etc.

Interface with External Functional Components

NCL are aimed at the specification of commands belonging to a command language. The user's goal is to execute the command(s). Therefore, the dialogue level must not only understand the naturally expressed command, but also execute it. In fact, this implies the definition and use of another Command Language between the NCLD System and the external functional components in charge of the execution of the commands.

3.5.1 Dialogue Commands

Both from an operational and a functional point of view, one of the main consequences of the model outlined is the generation of new commands at the dialogue level.

That is, the nature of the dialogue itself may allow the incorporation of new commands to the system as a whole, improving the overall performance, naturalness and flexibility.

A very simple example will illustrate the idea. Let's consider a telephone system that allows only the following set of commands (command language system):

- call(number)
- transfer(number)
- cancel-current-transfer

The use of a dialogue module with the user directory (where each entry has the number, name and office of a person) and a dialogue memory module (which stores the N-previous commands for each user, some new commands (dialogue commands) will appear.

First, the use of the directory may allow the user to call or transfer the calls using the name or the office number that identifies the intended telephone number:

- call(name)
- call(office)
- transfer(name)
- transfer(office)

But, if the system doesn't only consult the directory but is also able to store the last commands executed by each user, a new block of dialogue commands will appear at the dialogue level:

- retry-call
- retry-transfer
- consult-transfer-status
- consult-transfer-destination

It is worth noting that these two last sets of commands are seen as real commands from the user's point of view, both from a functional and operational perspective.

Nevertheless, there is a one-to-one correspondence between these dialogue commands and the low level command language system. In some situations, the dialogue level has to interpret and translate the dialogue command into one or more system commands (for instance, retry-call is translated into a call command incorporating the corresponding argument). And in other situations, the dialogue commands don't require the use of the command system at all (for instance, consult-transfer-status may obtain the result using only the user memory of the dialogue system).

3.6 Natural Command Language Dialogue Systems

A Natural Command Language Dialogue System (NCLDS) will be the computational module in charge of the analysis and manipulation of the corresponding Natural Command Language Dialogue model.

Figure 3.3 shows the role that NCLDSs play as the dialogue level interface between the user (through the NCLS) and the CLS.

3.7 A Natural Language Interface for Virtual Reality Systems

This section illustrates these ideas in a realistic scenario: a virtual reality system.

With this goal in mind, we have chosen the project “*A Natural Language Interface for Virtual Reality Systems*” [Everett, Wauchope & Pérez, 1996] developed by

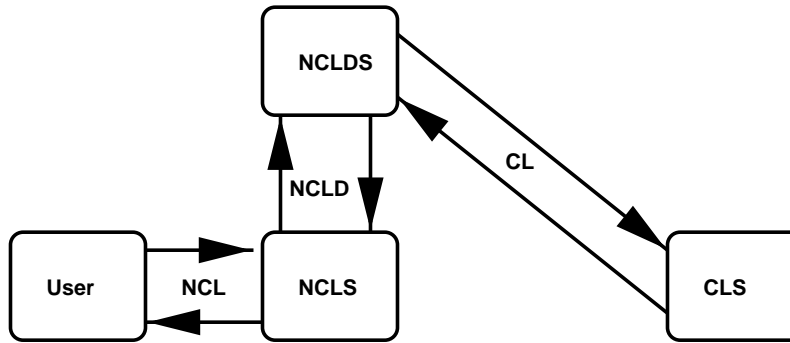


Figure 3.3: User (NCL) NCLS (NCLD) NCLDS (CL) CLS

the Navy Center for Applied Research in Artificial Intelligence. This project concentrated on a “prototype system designed to illustrate the potential usefulness of natural language understanding in speech interfaces to immersive 3D virtual environments”.

The main advantages of such a system where “... to enable the user to interact with the virtual environment system in a natural and intuitive way without removing his or her eyes from the display. It is especially useful in controlling things that do not have a physical presence in the environment, such as object scale, display characteristics, and time. It also provides a powerful means to access the knowledge that underlies the virtual environment by allowing the user to ask questions of the system. Using speech output in combination with speech recognition helps to avoid the use of textual display which can be difficult to read on immersive presentation equipment, and which can interfere with the user’s view of the ‘reality’ of the virtual world.”

From the point of view of the interests of this document, it is worth noting the main characteristics emphasized by the authors at the interface implementation level:

1. **Commands:** “The interface supports two classes of spoken input: commands and questions. The commands allow the user to control the playback of the simulation and its speed, as well as various display characteristics, such as viewpoint (‘Show me the top-down/out-the-window view’) and overlays (‘Display the map rings’). The user can also tell the system to hide or display individual objects (‘Show the Thunderbird’) or sets of ob-

jects ('Hide all the friendly aircraft that don't have missiles'). Commands are also used to manipulate time ('Run the simulation forward/backward'; 'Set the clock to zero'). In addition, the user can move from one object to another by name or by description, rather than by flying or pointing ('Put me on the Bladerunner'; 'Put me on the hostile ship'), or by specifying a particular location ('Move me to 23 degrees N, 40 degrees E; Increase my altitude to 4000 feet')."

2. **Questions:** "Questions allow the user to access information contained in the knowledge base that underlies the simulation. He/she can ask for information about the virtual world ('How many hostile ships are there?'), or about a specific object ('What is the Thunderbird's heading?'; 'What is my viewing altitude?') or set of objects ('Do any of the friendly ships have emitters on board?'). The user can also ask about the state of various aspects of the simulation ('Is the simulation running?'; 'What is the time increment?')."
3. **Reference and Discourse tracking:** "NAUTILUS [a general-purpose natural language processor] keeps a history of prior references and their denotations, which allows the use of anaphoric reference (pronouns like 'it' or 'them'). It also supports relative clauses ('All the ships that have missiles on board'), and elliptical follow-ups involving substitution noun phrases ('How about the Titanic?')."
4. **Feedback:** "User feedback is handled in one of two ways: changes in the graphical display in response to commands, and verbal answers output through the speech synthesizer in response to questions. Currently the answers consist primarily of yes/no and short noun phrases."

Next, we will try to analyze this scenario using the Natural Command Language Dialogue System framework.

1. The system as a whole has both information seeking (questions) and natural command language (commands) capabilities. So, for practical purposes is useful to think of a unique command language where commands may be classified into two main groups: consult commands (questions) and modify commands (proper commands). So, in order to be computationally effective, the system itself (in the previous scenario the virtual reality system) must be equipped with a command language, with commands like:

```
show_view(top_down)
show("Thunderbird")
move(me, 23, N)
consult(altitude)
consult(time_increment)
```

Of course, some of the proposed commands in the project description aren't low level commands, as the intended functionality of the command is obtained after a linguistic and discursive analysis (for instance: "Hide all the friendly aircraft that don't have missiles"). But, in any case, these upper-level commands may be expressed in a convenient programmable command language:

```
hide(consult(aircraft(type=friendly,missiles=0)))
```

With this command language, and the appropriate hardware interface (for instance, a keyboard) the user will be able to control (by means of questions and commands) the system. We have, therefore, a user and a Command Language System communicated by a Command Language.

2. But the use of a keyboard isn't suitable for a virtual reality environment. Instead of this, the authors suggest a hand and eye's free interface, and the best alternative is natural language: "Language, however, is ideally suited to abstract manipulations; it is also the most natural form of communication for humans, and does not require the use of one's hands or eyes."

So, a first goal is to substitute the formal and precise Command Language by a Natural Command Language. This will require the incorporation of a Natural Command Language System between the Natural Command Language used by the user and the formal Command Language supported by the computer (the system in general or the Command Language System according to our terminology).

In the virtual reality prototype, the authors used NAUTILUS (the Navy Automated Intelligent Language Understanding System), which is composed of four modules: the PROTEUS parser, the TINSEL case-frame interpreter, the FOCAL reference resolution module, and the FUNTRAN quantified-expression builder: "Building the prototype involved the creation of an application-specific dictionary and lexical semantics for NAUTILUS, a few minor extensions to its English grammar, and the development of two sets of code: one to translate the logical forms generated by NAUTILUS into messages for

the application software, and one to interpret these messages and instruct VIEWER to produce the appropriate actions or responses.” So, NAUTILUS plays here the Natural Command Language System functionality.

3. Nevertheless, “natural language understanding is not simply speech recognition, where each individual utterance maps to a specific command. In a natural language understanding system, a given sentence may have different meanings depending on the context, so a logical analysis of the utterance is required to determine the appropriate interpretation.”

Both consult and modify commands are extremely dependent on the context, which includes the running context (the simulation focus in the virtual reality environment) as well as the discourse context.

In the virtual reality interface under discussion, the dialogue capabilities of the natural language system were limited to simple cases of anaphora or ellipsis resolution. Obviously, the model lacks a specific dialogue control module.

This limitation is on the basis of two main deficiencies of the prototype:

- (a) The first one is explicitly recognized by the authors: “Currently the verbal responses to questions are limited to noun phrases, preceded by yes/no as appropriate. Addition of natural language response generation capabilities ... could significantly improve the interface and the flow of discourse.”
- (b) The second one isn’t mentioned, but must play a crucial role. It is related to the robustness of the system itself, conditioned by the fact that input is obtained using a speech recognition module. The question that arises is what happens with erroneously recognized input.

So, it seems reasonable to think of a dialogue module (in this case, a Natural Command Language Dialogue System) that will allow human-computer interactions (Natural Command Language Dialogues) defined for the specific set of tasks (commands), but improved with dialogue control operations, such as confirmation (when needed, depending on the characteristics of the input device), interaction, coordination, etc.

Chapter 4

Dialogue Moves in Natural Command Language Dialogues

Chapter 2 analyzed the main features of the three types of dialogue proposed: Information Seeking, Natural Command Language and Negotiative Dialogues. Next, chapter 3 described the elements, functions and structure of dialogue management systems based on Natural Command Languages. This chapter focuses on Natural Command Languages from the Information State Update approach, paying special attention to the specification and classification of Dialogue Moves for this type of dialogues.

Following the classification proposed in the Information State Theory of Dialogue Modelling [Traum *et al*, 1999], section 1 shows a description of the informational components of the information state.

In section 2 we propose a set of dialogue moves for Natural Command Languages dialogues. Our proposal is based on the corpus of dialogues of the telephone operator system (see Siridus Deliverable 3.1 [Torre, Amores & Quesada, 2000] and chapter 6 of this document) and examples from the Natural Interface for Virtual Reality Systems described in chapter 3.

The labels and representations which will be used in this section are merely indicative and implementation independent.

Section 3 contains the annotations of four dialogues, following the dialogue moves described in section 2. The first two dialogues belong to the Information Seeking Model (GoDiS and MIDAS), and the other two belong to the Natural Command Language model (TeleDelfos and Natural Language Unix Interface). Two consequences may be highlighted at this point. On the one hand, Natural Command Language Dialogue moves are sufficiently powerful to deal with Information Seeking Dialogues. Secondly, we have achieved a domain independent set of dialogue moves, which ensures the application of this model to other domains belonging to the Natural Command Language scenario.

The main differences between the information seeking and natural command language dialogue models appear in the control strategy in the application of the update rules. Therefore, section 4 is devoted to the description of the main characteristics of the Natural Command Languages Dialogues from the perspective of the control strategy.

4.1 Informational Components

- **Modeling Internal vs External aspects of the Dialogue:** As a consequence of their own nature, Natural Command Language Dialogues involve just two participants: the user and the machine. The first decision concerns whether we should model the participants' internal state, or more external aspects of the dialogue.

Since the goal of this type of dialogues is that the user have control over the execution of one or more commands by the machine, most dialogues exhibit a marked functional or operational tendency. In addition, the sequence of commands may sometimes lack some logical flow, or even be the consequence of obscure intentions during the dialogue.

For all these reasons, it seems reasonable to focus our model on the external aspects of dialogue. That is, it should be based more on what was said than what was in the minds of the participants when things were said.

- **Static and Dynamic Aspects:** One of the aspects which differentiates Natural Command Language Dialogues from Information Seeking Dialogues is the dynamic nature of the knowledge bases involved.

In an Information Seeking Dialogue, in which the system as a whole is viewed by the user as a repository of knowledge, knowledge bases (which

are called resources in TRINDIKIT) have a clear static character from the point of view of the user. That is, the data stored in these resources may be updated, but not by the user during its interaction with the system.

On the contrary, one of the main features of Natural Command Language Dialogues is the presence of a command execution system (what we called a Command Language System in chapter 3) which is capable of dynamically modifying the contents of external resources to the dialogue (during its interaction with the user), such as the knowledge bases associated to the domain.

4.2 Dialogue Moves

Following is a list of the dialogue moves which we have identified after the study of several scenarios belonging to Natural Command Language Dialogues, namely the *ATOS (Automatic Telephone Operator System)* dialogues corpus (described in detail in chapter 6), some examples from the description of the *A Natural Language Interface to Virtual Reality Systems* project, and some other examples obtained from artificially modelled possible natural language interactions with a database manager system.

First, we describe the list of dialogue moves we propose, along with a brief discussion of their motivation and some simple examples. Next, we have annotated the first 4 dialogues included in chapter 2. It is worth pointing out that the set of proposed dialogue moves for the Natural Command Language Dialogues allows the annotation and codification of Information Seeking Dialogues examples as well.

The table below summarizes the Dialogue Moves for Natural Command Languages classifying them into three groups:

	<i>Dialogue Moves (DM) in NCL</i>
Command-oriented DMs	askCommand specifyCommand informExecution
Parameter-oriented DMs	askParameter specifyParameter
Interaction-oriented DMs	askConfirmation answerYN askContinuation askRepeat askHelp answerHelp errorRecovery greet quit

We start with a group of dialogue moves which we have called *command-oriented*, in which we have included all dialogue moves dealing with command specification, such as askCommand, specifyCommand and informExecution.

4.2.1 askCommand

This move is related to those situations in which the system requests the user to specify a command or function to be performed.

For instance:

- *Specify a function, please.*
- *do you want a route?*

There are two possible notations for this dialogue move, depending on whether the move suggests or not a possible command:

- **askCommand** *Specify a function, please.*
- **askCommand(travelRoute)** *do you want a route?*

4.2.2 specifyCommand

This move takes place when a specific command or function has been selected. Often, a command specification may include one or more of its parameters.

For instance:

- *I want to place a call.*
- *flights to paris.*
- *I want to call 1 2 3 4 collect.*

The notation should include the command, and the set of instantiated parameters:

- **specifyCommand(phoneCall)** *I want to place a call.*
- **specifyCommand(fly,cityTo = paris)** *flights to paris.*
- **specifyCommand(collectCall,destination = 1234)** *I want to call 1 2 3 4 collect.*

4.2.3 informExecution

This corresponds to the actual execution of the command, which, of course, may change the state of the world and the information state of the participants. Once this dialogue move has finished, the participants know that the command has been executed. At a linguistic level, this move will be associated with utterances where the system acknowledges the execution of the task. In some situations, to improve naturalness, it may be an implicit move. In other cases, this move corresponds to the actual execution (as with a phone call).

- Example 1:
 - *Do you really want to call 1 2 3 4 collect?*
 - *yes, please.*

– *CALL COLLECT EXECUTION*

● Example 2:

– *Do you want the directory list ordered by name?*

– *Yes, please.*

– *DIRECTORY LIST.*

The notation should include the command plus all relevant parameters:

● **informExecution(collectCall,destination = 2134)** *CALL COLLECT EXECUTION*

● **informExecution(ls, directory = test, order = name)** *DIRECTORY LIST.*

Some commands will require not only acknowledgement of the execution as the previous examples illustrate, but the generation of the information requested. For instance, a database consult or a travel route query will generate some results that must be presented to the user in a specific dialogue move that will change the user's information state. In itself, this move interacts with other relevant issues such as presentation of the results by means of a text-to-speech system, choosing and selecting between possible results, or allowing the user to navigate through the set of results, etc.

Examples:

● *It will cost 2933 crowns*

● *Your route from cambridge to london*

From:	Cambridge, Gloucs
To:	London, GtLon
Total Distance:	111.7 Miles
Total Time:	2 hr 23 min

From a notational perspective, it is important to represent the command and the information obtained as the result of the command:

- **informExecution(fly, costs = 2933crowns)** *It will cost 2933 crowns*
- **informExecution(travelRoute, cityFrom = Cambridge, cityTo = London, totalDistance = 111.7 Miles, totalTime = 2 hr 23 min)** *Your route from cambridge to london*

From:	Cambridge, Gloucs
To:	London, GtLon
Total Distance:	111.7 Miles
Total Time:	2 hr 23 min

4.2.4 askParameter

As commands usually require specific parameters several dialogue moves control command completeness (mainly, instantiation of mandatory parameters). Two dialogue moves (`askParameter` and `specifyParameter`) are included in the group that we have called *Parameter-oriented Dialogue Moves*:

In the `askParameter` move, the system asks for the value of a specific parameter.

Examples:

- *What city do you want to go from?*
- *Do you want a return ticket?*
- *when do you go?*
- *Yes, who would you like to call?*

The notation should include the parameter (the command may be inferred from the dialogue context), and possibly the suggested value for the parameter if it was present (as in the last example):

- **askParameter(cityFrom)** *What city do you want to go from?*
- **askParameter(returnTicket)** *Do you want a return ticket?*

- **askParameter(when)** *when do you go?*
- **askParameter(destination)** *Yes, who would you like to call?*

4.2.5 specifyParameter

This move corresponds to the assignment of some value to one parameter.

For instance:

- *london.*
- *yes, as cheap as possible.*
- *I will start at nine am.*

In this case, the notation must include the parameter and its value, which, in some cases, may have a very complex structure.

- **specifyParameter(cityFrom = london)** *london.*
- **specifyParameter(returnTicket = yes)** *yes*, **specifyParameter(price = cheap)** *as cheap as possible.*
- **specifyParameter(when = 9am)** *I will start at nine am.*

4.2.6 askConfirmation

Finally, a third group includes those dialogue moves related to some intrinsic characteristics of the dialogue (*Interaction-oriented Dialogue Moves*), such as the confirmation of the command before its execution, sub-help dialogues, error recovery strategies (of special interest with spoken language systems), etc.

Once a command has been completed, some situations will require an explicit and/or implicit confirmation. In some applications, a confirmation may be required

for each command and parameter, in order to acknowledge each piece of information.

Example:

- *Do you really want to call 1 2 3 4 collect?*

It is important to specify the explicit part of the command that is being confirmed:

- **askConfirmation(collectCall,destination = 1234)** *Do you really want to call 1 2 3 4 collect?*

4.2.7 answerYN

This move represents the classical yes/no reply.

- *Yes.*
- *That's wrong.*

The current logical value will be stored as part of the dialogue move:

- **answerYN(yes)** *Yes.*
- **answerYN(no)** *That's wrong.*

4.2.8 askContinuation

Given the cyclical nature of command execution in this type of dialogues, each time a command has been completed (a sort of conversational game) there will follow a move asking for the continuation of the dialogue. This move could explicitly suggest a next command given the state of affairs and the last executed

commands. Thus, an **answerYN** affirmative reply would actually behave as a **specifyCommand** move.

Some examples are:

- *Do you want to continue?*

Whose representation would be:

- **askContinuation** *Do you want to continue?*

4.2.9 askRepeat

Any of the participants may request the other to repeat the last utterance, or even a specific parameter or command.

- Repeat example:
 - *Excuse me.*
 - *I want to call 1 2 3 4 collect.*
 - *Can you repeat the destination, please?*
 - *1 2 3 4.*

The askRepeat sentences in the previous sub-dialogue will be specified as follows:

- Repeat example:
 - **askRepeat** *Excuse me.*
 - *I want to call 1 2 3 4 collect.*
 - **askRepeat(destination)** *Can you repeat the destination, please?.*
 - *1 2 3 4.*

4.2.10 askHelp

This move shows up with a petition for help, which may be very general (i.e. functionality of the whole system), a specific command, or a specific parameter (i.e. the range of possible values), etc.

The following sub-dialogue illustrates this situation:

- Help sub-dialogue example:
 - *What are the available functions of the system?*
 - *You can place a phone call, consult your directory or send a message.*
 - *Well, how can I send a message?*
 - *To send a message, first you have to indicate the addressee of the message*
 - *That's better, but what types of addressees may I use?*
 - *The destination of a message may me a telephone number or a person in your directory.*

The annotation we suggest for these petitions are:

- Help sub-dialogue example (with askHelp notations):
 - **askHelp** *What are the available functions of the system?*
 - *You can place a phone call, consult your directory or send a message.*
 - **askHelp(sendMessage)** *Well, how can I send a message?*
 - *To send a message, first you have to indicate the addressee of the message*
 - **askHelp(destination)** *That's better, but what types of addressees may I use?*
 - *The destination of a message may me a telephone number or a person in your directory.*

4.2.11 answerHelp

This dialogue move corresponds to the reply to an **askHelp** move. Therefore, there are different modes depending on the type of help requested.

Next, we show the previous dialogue sample using this new dialogue move:

- Help sub-dialogue example (with askHelp and answerHelp notations):
 - **askHelp** *What are the available functions of the system?*
 - **answerHelp** *You can place a phone call, consult your directory or send a message.*
 - **askHelp(sendMessage)** *Well, how can I send a message?*
 - **answerHelp(sendMessage)** *To send a message, first you have to indicate the addressee of the message*
 - **askHelp(destination)** *That's better, but what types of addressees may I use?*
 - **answerHelp(destination)** *The destination of a message may be a telephone number or a person in your directory.*

4.2.12 errorRecovery

During the dialogue interaction, due to inconsistencies in the information provided, in what was understood by the participants or in what was recognized by a speech recognition system, there may arise situations in which the continuation of the dialogue is impossible.

In these situations, an **errorRecovery** dialogue move generated by a participant warns the other participant about the inconsistency detected, and suggests a restart from the last consistent state, or he/she may show some reservations about the item which may be causing the error or inconsistency.

The following example illustrates this situation:

- *I'm sorry. I don't understand which function you would like to perform. Let's*

start from the beginning. Could you tell me which function you would like to perform?

In this case, the recovery move has clearly identified the source of the error (the command was not recognized), and a repetition of the command is requested.

- **errorRecovery(command)** *I'm sorry. I don't understand which function you would like to perform. Let's start from the beginning. askCommand*
Could you tell me which function you would like to perform?

4.2.13 greet

This move describes the usual greeting operation:

- **greet** *Welcome to the system!*

4.2.14 quit

Likewise, this move represents the usual closing operation:

- **quit** *thanks, bye!*
- **quit** *Thank you for your visit!*

4.3 Annotations with NCL Dialogue Moves

The subsections below apply the Dialogue Move scheme above to the four pieces of dialogue introduced in Chapter 2.

4.3.1 Dialogue Moves in Dialogue 2.1: GoDiS

- *S(1)*: **greet** Welcome to the travel agency!.
- *U(1)*: **specifyCommand(fly,cityTo = paris)** flights to paris
- *S(2)*: **askParameter(cityFrom)** What city do you want to go from?
- *U(2)*: **specifyParameter(cityFrom = london)** london
- *S(3)*: **askParameter(returnTicket)** Do you want a return ticket?
- *U(3)*: **specifyParameter(returnTicket = yes)** yes, **specifyParameter(price = cheap)** as cheap as possible
- *S(4)*: **askParameter(departMonth)** What month do you want to leave?
- *U(4)*: **specifyParameter(departMonth = april)** april, please
- *S(5)*: **informExecution(fly,cost = 2933crowns)** It will cost 2933 crowns
- *U(5)*: **quit** thanks, bye!
- *S(6)*: **quit** Thank you for your visit!

4.3.2 Dialogue Moves in Dialogue 2.2: MIDAS

- *S(1)*: **askCommand** do you want a route?
- *U(1)*: **specifyCommand(travelRoute,cityTo = London)** I travel to London.
- *S(2)*: **askCommand(travelRoute)** well well well, do you want a route?
- *U(2)*: **answerYN(yes)** Yes.
- *S(3)*: **askParameter(departTime)** when do you go?
- *U(3)*: **specifyParameter(departTime = 9am)** I will start at nine am.
- *S(4)*: **askParameter(cityFrom)** where do you start?
- *U(4)*: **specifyParameter(cityFrom = Cambridge)** I like to start in Cambridge.
- *S(5)*: **wait** Please wait while I am calculating your rout ...

- *S(6)*: **informExecution(travelRoute, cityFrom = Cambridge, cityTo = London, totalDistance = 111.7 Miles, totalTime = 2 hr 23 min)** Your route from cambridge to london

From:	Cambridge, Gloucs
To:	London, GtLon
Total Distance:	111.7 Miles
Total Time:	2 hr 23 min

- *S(7)*: **quit** bye bye!

4.3.3 Dialogue Moves in Dialogue 2.3: TeleDelfos

- *S(1)*: **greet** Welcome to Delfos. **askCommand** Specify a function, please.
- *U(1)*: **specifyCommand(phoneCall)** good morning, I would like to place a call.
- *S(2)*: **askParameter(destination)** Yes, who would you like to call?
- *U(2)*: **answerYN(no)** no, **specifyCommand(collectCall,destination = 1234)** I want to call 1 2 3 4 collect.
- *S(3)*: **askConfirmation(collectCall,destination = 1234)** Do you really want to call 1 2 3 4 collect ?
- *U(3)*: **answerYN(no)** no, **specifyParameter(destination = 2134)** 2 1 3 4.
- *S(4)*: **askConfirmation(collectCall,destination = 2134)** Do you really want to call 2 1 3 4 collect ?
- *U(4)*: **answerYN(yes)** yes, please.
- *S(5)*: **informExecution(collectCall,destination = 2134)** CALL COLLECT EXECUTION
- *S(6)*: **askContinuation** Do you want to continue ?
- *U(5)*: **answerYN(no)** no thanks, **quit** goodbye.
- *S(7)*: **quit** Thanks for using the TeleDelfos system. Goodbye.

4.3.4 Dialogue Moves in Dialogue 2.4: Natural Language Unix Interface

- *S(1)*: **greet** Welcome to NL-Unix Interface. **askCommand** Specify a command, please.
- *U(1)*: **specifyCommand(ls,directory = test)** Please list the content of the test directory.
- *S(2)*: **askParameter(order,value = name)** Do you want the directory list ordered by name?
- *U(2)*: **answerYN(yes)** Yes, please.
- *S(3)*: **informExecution(ls,directory = test, order = name)** DIRECTORY LIST. Done. **askContinuation** What next?
- *U(3)*: **specifyCommand(rm,file = sample)** Delete the file sample.
- *S(4)*: **askConfirmation(rm,file = sample)** Do you want to delete the file sample?
- *U(4)*: **answerYN(yes)** yes, exactly.
- *S(5)*: **informExecution(rm,file = sample)** I've deleted the file sample. **askContinuation** Do you want to continue ?
- *U(5)*: **quit** no that's all.
- *S(6)*: **quit** Closing the NL-Unix Interface.

4.4 Control Strategy

The main features which may be identified at this level are:

- **User-guided**: The execution system and the overall system provide the user with a series of available commands from which he/she will be able to select one or more, provide its parameters, and confirm its execution.

Therefore, the user is in control of the system; he/she may select and change his/her mind at any point, as regards the command itself or the values of its parameters.

On the other hand, the dialogue system takes over the control in the identification of the parameters and during the confirmation and execution of the command.

- **Help Sub-dialogues:** The availability of multiple commands, each with its own number and type of arguments (both obligatory and optional), introduces a level of complexity in the control strategies higher than that found in Information Seeking Dialogues.

Therefore, in order to guarantee the success in the execution of the command by the system, it is crucial to have at our disposal a sophisticated system of help subdialogues which provides the user with detailed information of the state of affairs at any time.

In so far as the user perceives the system as a functional module aimed at the execution of commands, the help supplied by the system should neglect the characteristics and the context of the dialogue itself, and focus on the exact and active state of affairs at the consultation time, as well as on the possibilities of continuation at that point.

- **Control Stages (Cyclic Structure):** This type of dialogues exhibit a clear cyclical structure, whose main phases, with possible overlappings, are:
 - **Command detection:** First and foremost, we must be able to identify the command which the user intends to be performed, among those offered by the system. At this level, it should be pointed out that the dialogue manager should be able to warn the user if he/she tries unsupported commands.
Also related to this phase is the possibility of suggesting alternative commands which may replace the selected command as a way of improving or suggesting alternative execution strategies, as is common in Negotiative Dialogues.
 - **Command completion:** Commands may require parameters which may modify their mode of execution. Therefore, the dialogue management system must be able to check the completeness of the command selected by the user and request from him/her all the required and yet unspecified arguments.
We should also take into account that this phase may be inexistent from the point of view of the dialogue interaction between the user and the

system, since the user may have specified all required parameters as part of his/her request. This should gradually be more common in users familiarized with the system.

It should also be pointed out that this phase may be critical for novel or inexperienced users, which reinforces the need of an adequate subsystem of help dialogues at this level.

- **Command confirmation:** A specific confirmation phase is required since executing some commands may modify external resources.

From the dialogue management standpoint, it will be important to introduce both explicit and implicit confirmation strategies, which make the interaction more flexible and natural.

In addition, when dialogue management systems are used by means of spoken interfaces confirmation becomes a relevant aspect which avoids the execution of misrecognized commands.

- **Output generation:** Executing some commands may be oriented towards obtaining some (possibly complex) information. In this case, and especially if the interaction takes place through spoken interfaces, it is crucial to have at our disposal output subdialogues which display the result of the query and allow him/her to navigate through the information provided by the system.
- **Dialogue continuation:** Finally, the usual cyclic structure of the dialogue ends by means of a specific phase which allows the user to link two command execution cycles. From a practical point of view, this continuation phase will be linked to the initial phase of the next cycle (command detection and completion) or exiting.
- **Dialogue exit:** Depending on the specific characteristics of the command language, it may be necessary to include in this phase the execution of some operations pertaining to the management of external resources (closing access to resources, committing permanent changes, storing general management parameters such as personalized memory variables, etc.) or those pertaining to the user, such as providing a summary of executed commands, consumption of resources, etc.

- **Connection with external resources:** The relevance that external resources (databases and knowledge associated with consultation) have in Natural Command Language Dialogues justifies the incorporation of sophisticated control mechanisms in the interfaces with those resources (following TRINDIKIT terminology).

Specifically, it will be necessary to have access to commit and rollback mechanisms and to personalization strategies in the use of environment commands (which should be transparent to the user in charge of storing and recovering such environment), etc.

- **Updating Algorithm:** The most relevant aspect at this level has to do with dialogue recovery strategies. Given an incomprehensible input which does not make sense in the current dialogue configuration, nor according to the expectations generated for that configuration, or which substantially modifies the logic of discourse, it is necessary to activate strategies which help a flexible recovery of the dialogue, reminding the user of the current situation.
- **Selection Algorithm:** The strategy for the selection of updating rules is of special relevance in systems geared towards multiple tasks (commands), since the similarity in the linguistic and functional structure of some commands might lead to an erroneous interpretation of dialogue moves. Therefore, the selection algorithm must take into account the semantic content of dialogue moves.

Chapter 5

Comparing Natural Command Language Moves with Other Move Coding Schemes

This chapter attempts to compare the taxonomy of NCL Moves presented in chapter 4 with other move coding schemes. As it was previously done in Trindi, we will try to point out the similarities and differences between our coding scheme and that of other projects and scenarios such as TRAINS, DRI, MapTask, LINLIN2, GBG-IM both at the level of scheme layers and move taxonomies.

5.1 Scheme Layers

As regards scheme layers, we have classified our moves into three groups: Command-oriented, Parameter-oriented and Interaction-oriented.

Table 2.1 in [Cooper *et al*, 1999] *Rough Impression of relations between scheme layers* may be reused to some extent in order to accommodate our scheme layers as follows (Table 5.1):

LINLIN2	MapTask	DRI	TRAINS	GBG-IM	NCL
—	Game		Argumentation Acts		
Type	Move	Forward-Looking Backward-Looking	Core Speech Acts		Command-oriented Parameter-oriented
		Signal Understanding	Grounding Acts	Feedback function	Interaction-oriented
—	—	—	Turn-taking	Turn characteristics	
Discourse management	—	Conventional	—	—	
Topic	—	Information level	—	—	—
—	—	Communicative Status	—	—	—

Table 5.1: Rough impression of relations between scheme layers

5.2 Move Taxonomies

Similarly, Tables 2.2 and 2.3 in [Cooper *et al*, 1999] may guide us in comparing NCL moves with previous coding schemes.

LINLIN2	MapTask	DRI	TRAINS	GBG-IM	NCL
<i>Initiative</i>	<i>Initiating moves</i>	<i>Forward Looking Function</i>	<i>Core speech acts</i>	—	—
Update	Explain	Statement Assert Reassert Other	Inform		
Question	Query-yn Query-w Check Align	Info-request	YNQ WHQ		askCommand askParameter askConfirmation askHelp
—	Instruct	<i>Influencing-addressee-future-action</i> Action-directive Open-Option	Request Suggest		specifyCommand
	—	<i>Committing-speaker-future-action</i> Offer Commit	Offer		—
	—	Explicit-performative	Promise		informExecution
	—	Exclamation	—		—
Response (Answer)	<i>Response moves</i>	<i>Backward Looking Function</i>	<i>Core speech acts</i>	—	—
	Reply-y, Reply-n, Reply-w, Clarify	Answer	Eval		answerYN answerHelp specifyParameter
	—	<i>Agreement</i> Accept Accept-part Maybe Reject Reject-part Hold	Accept Reject	+Accept-content -Accept-content	—
—	Ready ???	—	—	—	—

Table 5.2: Rough impression of relations between move taxonomies, pt. 1

Given that an all-encompassing comparison is beyond the scope of this report, we have limited our comparison to two well-known coding schemes, namely DRI and TRAINS.

DRI has been adopted by Trindi as a base standard model, and TRAINS is similar to the NCL scenario insofar as both have been designed for a human-machine dialogue application.

5.3 A Comparison between DRI, TRAINS and NCL Moves

What follows is a detailed comparison between NCL moves and those proposed by DRI and TRAINS. We first describe an NCL move and then discuss how it relates to other TRAINS and DRI moves.

LINLIN2	MapTask	DRI	TRAINS	GBG-IM	NCL
—	<i>(Response moves)</i>	<i>Understanding</i>	<i>Grounding</i>	<i>Feedbackfunction</i>	—
	—	—	ReqAck	Elicit FB	
	—	—	ReqRepair	—	
	(Acknowledge)	—	—	+Accept-com-act	
	—	—	—	-Accept-com-act	
	(Acknowledge)	<i>Signal-understanding</i>	Ack	+Understanding	
	—	Acknowledge	—	—	
	—	Repeat-rephrase	—	—	
	—	Completion	—	—	
	—	Signal-Non-Understanding	—	-Understanding	
	Acknowledge	—	—	+Perception	
	—	(Signal-non-und.)	—	-Perception	
—	—	—	+Contact		
—	(Signal-non-und.)	—	-Contact		
—	Correct-Misspeaking	Repair	—	askRepeat	
—	—	Initiate	—	errorRecovery	
—	—	Continue	—		
—	—	Cancel	—		
—	—	—	<i>Turn-taking</i>	<i>Turn management</i>	—
			take-turn	Turn acceptance	
			keep-turn	Turn holding	
			release-turn	Turn closing	
			assign-turn	—	
<i>Discourse management</i>	—	<i>Conventional</i>	—	—	—
Opening		Opening			
Continuation		—			
Closing		Closing			
—	—	—	—	—	greet
—	—	—	—	—	askContinuation
—	—	—	—	—	quit

Table 5.3: Rough impression of relations between move taxonomies, pt. 2

- **askCommand**

This move is related to those situations in which the system requests the user to specify a command or function to be performed. It is, therefore, a Forward-Looking Function of type Info-Request in the DRI scheme, since it "creates an obligation for the hearer to provide information, using any form of communication" [Allen & Core, 1997] (page 11).

This move is usually in the form of a WH- or a Y/N Question, according to the TRAINS coding scheme. As a Grounding Act Type, it corresponds to an Initiate Act, since it *"usually corresponds to the (first utterance in the) presentation phase of a top level Contribution."* [Traum & Hinkelman, 1992] (page 5)

- **askParameter**

In this move, the system requests the value of a specific function parameter. In this sense, it is very similar to the previous **askCommand** move.

- **askHelp.**

This move may be seen as a specific subtype of the previous two moves, in the sense that the speaker is requesting information about how to proceed with a particular telephone function. However, it could also be seen as Committing the speaker future action since it *"potentially commits the speaker (in varying degrees of strength) to some future course of action."* [Allen & Core, 1997] (p. 13)

- **askConfirmation.**

This move requests explicit confirmation from the speaker, typically expressed in the form of a Yes/No question. It is similar to the Check move in the MapTask, and to the ReqAck Act in the TRAINS taxonomy (*"Attempt to get the other agent to acknowledge the previous utterance."* [Traum & Hinkelman, 1992] (p. 6)

- **specifyCommand.**

This move takes place when a specific command has been selected by the speaker. It is, therefore, influencing the addressee future action in the form of an action-directive move, since it *"obligates the listener to either perform the requested action or communicate a refusal or inability to perform the action."* [Allen & Core, 1997] (p. 12). It is thus similar to the Core speech act Request in TRAINS.

- **specifyParameter.**

This move may look similar to the previous one at first sight, since it is used to assign some value to a specific function parameter. However, they are usually answers to previous (or defaulted) **askCommand** moves. In this sense, they are similar to Clarify moves in MapTask or Answer in DRI, since they are Backward-Looking Functions.

In the TRAINS taxonomy, this move would be similar to an Inform Core Speech Act.

- **informExecution.**

This move corresponds to the actual execution of the command. In a sense, it may be seen as an explicit performative whereby the system actually performs the requested action (telephone function) while saying it, i.e. the speaker (system) commits itself to executing a specific action. In a sense it is a type of implicit acknowledgement from the system, and thus similar to the Acknowledge Act in TRAINS.

- **answerYN.**

This is the classical reply to a Yes/No question, present in most coding schemes as Reply-y, Reply-n, Eval, Answer, etc. Note that here we include DRI and TRAINS Accept and Reject Agreement moves. From a grounding perspective, they also show a type of implicit acknowledgement.

The dialogue manager must then disambiguate the reply and take it as a neutral yes/no answer or as an accept/reject move.

Finer moves such as Accept-part and Reject-part are also possible in our scenario. However, no specific move has been adopted here. Instead, they may be captured as combinations of rejecting/accepting plus specifying something new.

- **answerHelp.**

Providing explicit information on how to perform a command, or what parameters are required, may be seen as an Answer to an **askHelp** move, but also as Influencing the addressee future action in an open-option move, since it *"suggests a course of action but puts no obligation on the listener."* [Allen & Core, 1997] (p. 12)

This may be seen very clearly if the reply contains, for example, all possible functions to be performed by the system in our scenario. In TRAINS, this move corresponds to two acts, depending on whether help is provided about the overall functionality of the system (in this case it would be similar to a Suggest Speech Act) or an Inform Act if the system provides information about a specific function.

- **askRepeat.**

With this move, any participant may request the other to repeat the last utterance, or a specific parameter or command. It is, therefore, a case of partial

Signal–Non–understanding in DRI, and identical to the ReqRepair move in TRAINS.

- **errorRecovery.**

This move marks cases in which the continuation of the dialogue is impossible due to a high degree of misunderstanding. It may be seen as an extreme case of Signal–Non–Understanding in DRI, and similar to a Cancel move in TRAINS.

- **greet.**

Is similar to the Opening move in DRI, and to the Initiate Act in TRAINS.

- **quit.**

Is similar to the Closing move in DRI.

- **askContinuation.**

This move suggests starting a new dialogue game. Strictly speaking, it is a subtype of a yes/no question which marks a transition from one game to another. It is therefore similar to the Continuation Discourse management move in LINLIN2, with no explicit counterpart in TRAINS or DRI.

Chapter 6

Natural Command Language Dialogue Moves for the Spanish Telephone Command Dialogue Corpus

This chapter applies the model of dialogue moves proposed in chapter 4 to a set of dialogues belonging to the Spanish Telephone Command Dialogue (STCD) corpus.

The main features of this scenario have been discussed in document *D3.1, User Requirements on a Natural Command Language Dialogue System* [Torre, Amores & Quesada, 2000], which also includes a set of representative dialogues in this corpus as well as a detailed description of the expected functionality of the system and the supported commands.

6.1 Annotating the Spanish Telephone Command Dialogue Corpus

This section contains the annotation of the dialogue number 10 shown in chapter 6 of [Torre, Amores & Quesada, 2000], using the dialogue move scheme described

above.

- *S(1)*: **greet** Welcome to the system. **askCommand** Which function would you like to perform?
- *U(1)*: **specifyCommand(phoneCall)** I want to place a call **SILENCE specifyCommand(phoneCall)** place a call.
- *S(2)*: **askParameter(destination)** Who would you like to call?
- *U(2)*: **specifyParameter(destination = 9898)** Extension nine eight nine eight.
- *S(3)*: **informExecution(phoneCall,destination = 9898)** I'll put you through to extension nine eight nine eight.
- *U(3)*:
- *S(4)*: **askContinuation** Would you like to perform any other function?
- *U(4)*: **specifyCommand(phoneCall)** I want to place a call.
- *S(5)*: **askParameter(destination)** Who would you like to place a call?
- *U(5)*: **answerYN(no)** No, **specifyCommand(collectcall)** I want to place a collect call.
- *S(6)*: **askParameter(destination)** Who would you like to call?
- *U(6)*: **answerYN(no)** No, **specifyCommand(sendMessage)** I want to send a message.
- *S(7)*: **askParameter(destination)** If you want to place a collect call, you must provide a telephone number or a name.
- *U(7)*: **specifyCommand(sendMessage)** I want to send a message.
- *S(8)*: **errorRecovery** I'm sorry. I don't understand which function you would like to perform. Let's start from the beginning. **askCommand** Could you tell me which function you would like to perform?
- *U(8)*: **specifyCommand(sendMessage)** I want to send a message.
- *S(9)*: **askParameter(destination)** Who would you like to send a message?

- *U(9)*: **specifyParameter(destination = JuanMedinaRobles)** To Juan Medina Robles.
- *S(10)*: **answerHelp(sendMessage)** Please, record your message after the tone. Pi.
- *S(11)*: **askConfirmation(sendMessage,destination = JuanMedinaRobles)** Sending the message to Juan Medina Robles.
- *U(10)*: **answerYN(no)** No, **specifyParameter(destination = JuliaMedinaRobles)** Julia Medina Robles.
- *S(12)*: **askConfirmation(destination = JuliaMedinaRobles)** Julia Medina Robles?
- *U(11)*: **answerYN(yes)** Yes.
- *S(13)*: **informExecution(sendMessage,destination = JuliaMedinaRobles)** Sending the message to Julia Medina Robles.
- *U(12)*:
- *S(14)*: **askContinuation** Would you like to perform any other function?
- *U(13)*: **specifyCommand(collectCall)** I want to place a collect call.
- *S(15)*: **askParameter(destination)** Who would you like to call?
- *U(14)*: **specifyParameter(destination = 913379894)** To the number nine one three three seven nine eight nine four.
- *S(16)*: **informExecution(collectCall,destination = 913379894)** I'll put you through to the number nine one three three seven nine eight nine four.
- *U(15)*: **answerYN(no)** No, **specifyParameter(destination = 913379898)** nine one three three seven nine eight nine eight.
- *S(17)*: **askConfirmation(destination = 913379898)** Number nine one three three seven nine eight nine eight?
- *U(16)*: **answerYN(yes)** Yes.
- *S(18)*: **askConfirmation(collectCall,destination = 913379898)** I'll put you through to the number nine one three three seven nine eight nine eight.
- *U(17)*:

- *S(19)*: **informExecution** Would you like to perform any other function?
- *U(18)*: **answerYN(yes)** Yes.
- *S(20)*: **askCommand** Then, specify a function or ask for help, if you dont know the functions.
- *U(19)*: **specifyCommand(sendMessage)** I want to send a message.
- *S(21)*: **askParameter(destination)** Who would you like to send a message?
- *U(20)*: **specifyCommand(phoneCall, destination = JuanPerezGonzalez)** I want to place a call to Juan Perez Gonzalez.
- *S(22)*: **informExecution(phoneCall, destination = JuanPerezGonzalez)** I'll put you through to Juan Perez Gonzalez.
- *U(21)*:
- *S(23)*: **askContinuation** Would you like to perform any other function?
- *U(22)*: **answerYN(yes)** Yes.
- *S(24)*: **askCommand** Then, specify a function or ask for help, if you dont know the functions.
- *U(23)*: **specifyCommand(retry)** I want to try the last call again.
- *S(25)*: **informExecution(retry)** I'll try to reconnect you.
- *U(24)*:
- *S(26)*: **askContinuation** Would you like to perform any other function?
- *U(25)*: **answerYN(no)** No.
- *S(27)*: **quit** Thank you for using the system.

Bibliography

- [Alexandersson *et al*, 1998] Alexandersson, J., B. Buschbeck-Wolf, T. Fujinami, M. Kipp, S. Koch, E. Maier, N. Reithinger, B. Schmitz & M. Siegel. (1998) *Dialogue Acts in VERBMOBIL-2*. DFKI Saarbrücken and TU Berlin, Report 226, July 1998.
- [Allen & Core, 1997] Allen, J. and Core, M. (1997). DAMSL: Dialog Act Markup in Several Layers. Draft contribution for the Discourse Resource Initiative.
- [Albesano *et al*, 1997] Albesano, D., Baggia, P., Gemello, R., Gerbino, E., and Rullent, C. (1997) A robust system for human-machine dialogue in a telephony-based application. *Journal of Speech Technology*, 2(2), 99-110.
- [Allwood *et al*, 1994] Allwood, J., Nivre, J., and Ahlén, E. (1994) Semantics and Spoken Language: Manual for Coding Interaction Management. Report from the HSF project *Semantik och talspråk*.
- [Andry *et al*, 1990] Andry, F., E. Bilange, F. Charpentier, K. Choukri, M. Ponomalé, & S. Soudoplatoff. (1990) *Computerised simulation tools for the design of an oral dialogue system*. Selected Publications, 1988-1990, SUNDIAL Project (Esprit P2218). Commission of the European Communities.
- [Aust & Oerder 95] Aust, H. & M. Oerder. 1995. Dialogue Control in Automatic Inquiry Systems. In Andernach, J. A., S. P. van der Burgt & G. F. van der Hoeven. eds. *Proceedings of the 9th Twente Workshop on Language Technology*. University of Twente, Netherlands.
- [Austin, 1962] Austin, J.L. (1962). *How to do Things with Words*. Harvard University Press, Cambridge, MA.
- [Cooper, 1998] Cooper, R. (1998). Information States, Attitudes and Dialogue, In *Proceedings of ITALLC-98*. Also available as GPCL 98-5 at <http://www.ling.gu.se/publications/GPCL.html>

- [Cooper & Larsson, 1999] Cooper, R., and Larsson, S. (1999). Dialogue moves and information states. In *Proceedings of the Third IWCS*, Tilburg.
- [Cooper *et al*, 1999] Cooper, R., Larsson, S., Matheson, C., Poesio, M., and Traum, D. (1999). Coding Instructional Dialogue for Information States. Deliverable D1.1, Trindi Project.
- [Everett, Wauchope & Pérez, 1996] Everett, S., Wauchope, K., and Pérez, M. A. (1996). A Natural Language Interface for Virtual Reality Systems. Navy Center for Applied Research in Artificial Intelligence, US Naval Research Laboratory, Washington, DC. <http://ftp.aic.nrl.navy.mil/severett/vr-report/title.html>
- [Kamp, 1981] Kamp, H. (1981) A theory of truth and discourse representation. In Groenendijk, J., Jansen, T. & Stockhof, M. eds. *Formal methods in the study of language*. Amsterdam: Mathematical Centre tracts 135.
- [Kamp & Reyle, 1993] Kamp, H. & Reyle, U. (1993) *From Discourse to Logic*. Dordrecht: Kluwer.
- [Poesio *et al*, 1999] Poesio, M., R. Cooper, S. Larsson, D. Traum & C. Matheson. (1999) Annotating Conversations for Information State Updates. In *Proceedings of the Amsteloog99 Workshop on the Semantics and Pragmatics of Dialogue*. Amsterdam University, May 1999.
- [Torre, Amores & Quesada, 2000] Torre, D., J. G. Amores and J. F. Quesada. (2000) *User Requirements on a Natural Command Language Dialogue System*. Deliverable 3.1. Siridus project.
- [Traum & Hinkelman, 1992] Traum, D., and Hinkelman, E. A. (1992). Conversation acts in task-oriented spoken dialogue. *Computational Intelligence*, 8(3). Special Issue on Non-literal Language.
- [Traum *et al*, 1999] Traum, D., Bos, J., Cooper, R., Larsson, S., Lewin, I., and Matheson, C. (1999). A model of dialogue moves and information state revision. Deliverable D2.1, Trindi Project.